TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUSSI EERIO

GENEERISEN KORTTIPELIMOOTTORIN SUUNNITTELU

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 6. helmikuuta 2014

TIIVISTELMÄ

JUSSI EERIO: Geneerisen korttipelimoottorin suunnittelu
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua
Lokakuu 2014
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: Geneerisyys, korttipeli, liitännäinen, liitännäispohjainen sovelluskehitys

Erilaiset korttipelit ovat jo pitkään olleet suosittuja ympäri maailmaa. Tämän vuoksi ei ole yllättävää, että niistä on vuosien mittaan tehty myös digitaalisia versioita. Lähes jokaisen tietokoneen mukana tulevan pasianssin lisäksi myös monet erilaiset keräilykorttipelit ovat saaneet omat digitaaliset vastineensa. Näistä tunnetuin on Magic: The Gathering, josta on toteutettu useita erilaisia digitaalisia versioita sekä alkuperäisen korttipelin julkaisijan että fanien toimesta.

Keräilykorttipelien digitalisoinnissa on monenlaisia ongelmia. Niiden perussäännöt ovat yleensä pohjimmiltaan yksinkertaiset, mutta monet kortit joko lisäävät, muuttavat tai kiertävät näitä perussääntöjä. Yksi toteutuksen haasteista on näiden poikkeuksien huomioiminen. Keräilykorttipelit ovat myös hyvin monimuotoisia. Jokaisella keräilykorttipelillä on omat korttityyppinsä ja niiden toimintaan liittyvät sääntönsä, minkä vuoksi suurin osa korttipelimoottoreista tukee vain yhtä korttipeliä. Korttipelin sääntöjen toteutuksen lisäksi myös korttipelien pelattavuuden takaaminen tarjoaa omat haasteensa. Kasvotusten pelatessa pelaajat voivat siirtyä vaiheesta toiseen hyvin nopeasti. Keskeisenä ongelmana on pelin sujuvuuden ja salaisen informaation säilyttämisen välinen ristiriita.

Tässä diplomityössä tavoitteena oli suunnitella geneerinen korttipelimoottori, jonka päälle olisi mahdollista rakentaa toteutus periaatteessa mille tahansa keräilykorttipelille. Korttipelimoottorin suunnittelussa päädyttiin hyödyntämään liitännäispohjaista sovelluskehitystä. Liitännäisiä hyödyntämällä sovellus voidaan jakaa geneeriseen, kaikille korttipeleille yhteiseen toteutukseen ja sen päällä toimiviin korttipelikohtaisiin toteutuksiin. Osana suunnittelutyötä lähdettiin kehittämään prototyyppiä geneerisen osion tarvitsemien ominaisuuksien määrittelemiseksi. Prototyyppi sisälsi geneerisen korttipelimoottorin ja sen päällä toimivan yksinkertaisen korttipelin toteutukset.

Työn tuloksena saatiin toteutettua proof-of-concept -prototyyyppi geneerisestä korttipelimoottorista ja sen päällä toimivasta korttipelitoteutuksesta. Prototyypin perusteella voidaan todeta, että liitännäispohjainen sovelluskehitys toimii hyvin geneerisen korttipelimoottorin toteutuksessa. Työn tulosten perusteella on hyvä jatkaa työn laajuuden ulkopuolelle jätettyjen osien suunnittelua ja geneerisen korttipelimoottorin toteutusta.

ABSTRACT

JUSSI EERIO: Designing a Generic Card Game Engine
Tampere University of Technology
Master of Science Thesis, 45 pages
October 2014
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen

Keywords: Genericity, card game, plugin, plugin-based development

Card games have enjoyed widespread popularity for decades. Due to their popularity it is hardly surprising that many card games have inspired a variety of digital adaptations. This applies to both the more traditional solitaires as well as collectable card games and trading card games such as Magic: The Gathering. The most popular card games have inspired multitudes of both official adaptations as well as fan-made creations.

Digitalizing collectable card games is not straightforward. While the basic rules of most card games are simple, many cards bend, break, or ignore these rules. Even though all card games share some similarities, they also have their unique card types and rules that are subject to human interpretation. Combined with the aforementioned bending of these unique rules, it is no wonder that the majority of card game engines only support one specific game and often only in a simplified form. However, rules are not the only problematic factor. The card game engine also needs to balance between keeping the progression of the game smooth and not revealing any hidden information. For example, when playing face to face, the players tend to skip phases in which they do not wish to take actions. A card game engine has to both go through every phase of a turn and give each player a chance to act in them regardless of the options available to them.

The goal of this thesis was to design a generic card game engine capable of supporting practically any collectable card game. A variation of a plug-in-based application design was chosen as the basis for the architecture. The use of plugins allows the card game engine to be split into generic parts that are used by all card games and game-specific parts that implement the rules of a specific game. In addition, a prototype of the engine was created as part of the design process to help determine the required features and to study the feasibility of plugins in developing a generic card game engine. The prototype included an implementation of both the generic card game engine as well as a simple card game to run on it.

Based on the proof-of-concept prototype, the plug-in-based application design was deemed a good fit for the creation of a generic card game engine. This thesis lays a good foundation for the design and creation of a complete card game engine in the future. As such the approach of this study can be considered a success.

ALKUSANAT

Tämä työ sai alkunsa omasta harrastuspohjastani ja kiinnostuksestani korttipelejä ja etenkin niiden kehittämistä kohtaan. Työn tuloksien pohjalta on tulevaisuudessa tarkoitus kehittää sekä omaan että muiden käyttöön sovellus, jota voi käyttää apuna sekä korttipelien kehittämisessä että testaamisessa.

Työn tarkastajana ja ohjaajana toimi Tommi Mikkonen, jota haluan kiittää avusta ja opastuksesta pitkän työrupeaman aikana. Lisäksi haluan kiittää ystäviä ja perhettä, joilta olen saanut tukea ja rohkaisua työn loppuun saattamiseksi.

Tampereella, 9.9.2014

Jussi Eerio

SISÄLLYSLUETTELO

1. JOHDANTO

Keräilykorttipelit ovat suosittua ajanvietettä ympäri maailmaa. Niiden menestyksen vuoksi ei ole yllättävää, että niistä on ajan myötä tehty myös monenlaisia adaptaatiota digitaalisille alustoille. Näitä adaptaatioita on tehty niin korttipelien alkuperäisten julkaisijoiden kuin fanien toimesta. Osa fanien kehittämistä adaptaatioista on aloittanut elämänsä korttipakkojen suunnitteluun tarkoitettuna tietokantoina ja kasvaneet siitä varsinaisiksi korttipelimoottoreiksi.

Korttipelien monimuotoisuuden seurauksena niitä pyörittävät pelimoottorit voidaan jakaa karkeasti kahteen ääripäähän: niihin, jotka tukevat vain yhtä korttipeliä kaikkine sääntöineen, sekä niihin, jotka antavat vaihtelevan tasoisen tuen useammalle korttipelille ottamatta kantaa yhdenkään sääntöihin. Jälkimmäisen ääripään edustajat ovat enemmänkin virtuaalisia pelipöytiä, jotka tukevat korttipakan simuloimista ja tarjoavat kourallisen toimintoja, jotka helpottavat korttipelin pelaamista. Varsinaiset korttipelin mekaniikat jätetään käyttäjien tulkittaviksi. Yksinkertaisimmillaan tämä tarkoittaa, että pelaaja näyttää vastustajalleen kortin, jonka aikoo pelata, ja osoittaa kortin mahdollisen kohteen sovelluksen korostustyökalulla ja – molempien pelaajien mahdollisten reaktioiden jälkeen – pelaajat yhdessä muuttavat pöydän tilanteen vastaamaan pelattujen korttien efektien jälkeistä tilannetta.

Käytännössä sääntöjä tukevia sovelluksia ei löydy kovin monelle alun perin fyysisillä korteilla pelattavaksi tarkoitetulle korttipelille ja myös suuri osa lähempänä virtuaalipöytää olevista korttipelimoottoreista on pääasiallisesti suunniteltu tukemaan Magic: The Gathering -korttipeliä (Wizards of the Coast, 1993). Tämän myötä virtuaalipöydät harvoin tarjoavat muille korttipeleille hyödyllisiä työkaluja ja niiden pelattavuus on suoraan verrannollinen niiden yhtäläisyyksiin Magicin kanssa. Tämän lisäksi pelin pelaaminen tuntemattoman kanssa on työlästä, sillä monimutkaisempien toimintojen välittäminen vastustajalle toimii pääasiassa sovelluksiin usein sisäänrakennetun chatin kautta.

Edellä mainittujen ääripäiden lisäksi on olemassa myös täysin digitaalisiksi suunniteltuja korttipelejä, joiden kehityksessä on alusta alkaen otettu huomioon digitaalisen toteutuksen rajoitukset ja toisaalta myös edut. Tämän hetken tunnetuin esimerkki tästä kategoriasta on luultavasti Hearthstone: Heroes of Warcraft -nettikorttipeli (Blizzard Entertainment, 2014). Tämän tyyppiset korttipelit yleensä karsivat pelaajien vuorovaikutusta monin eri tavoin verrattuna kasvotusten pelattaviin korttipeleihin verkkopelaamisen sujuvuuden parantamiseksi. Hearthstone esimerkiksi rajoittaa pelaajan toiminnan täysin pe-

laajan omiin vuoroihin. Äärimmäisenä esimerkkinä rajoituksista voidaan antaa korttipelit, joissa varsinaisen korttipakalla pelaamisen sijaan pelaajan vastuulle jää vain korttipakan suunnittelu: itse pakalla pelaaminen on automatisoitu.

Tämän työn tavoitteena on suunnitella ja toteuttaa prototyyppi geneerisestä korttipelimoottorista, jonka päälle voidaan rakentaa täysi tuki erilaisten korttipelien pelaamista varten. Prototyypin tarkoituksena on testata liitännäisten ja liitännäispohjaisen sovelluskehityksen hyödyllisyyttä geneerisen korttipelimoottorin toteutuksessa ja toimia pohjana tulevaisuuden jatkokehitykselle. Itse korttipelimoottorin pääasiallisena tarkoituksena tulee olemaan korttipeli-ideoiden testaaminen sekä korttipelien suunnittelu ja mahdollinen esittely verkon ylitse. Tässä työssä keskitytään vain digitaalisina julkaistujen korttipelien sijaan perinteisempiin keräilykorttipeleihin.

Työn puitteissa esitellään Magic: The Gathering -korttipelin säännöt perustasolla. Magicin historiaa keräilykorttipelien uranuurtajana hyödynnetään työssä käyttämällä sen sääntöjä ja termejä yleistyksinä tekstin luettavuuden ja ymmärrettävyyden parantamiseksi.

Luvussa 2 käsitellään korttipelejä yleisesti, käydään läpi korttipelien jaottelua eri tyyppeihin ja esitellään yleisellä tasolla korttipelien tavoitteita. Luvussa 3 tarkastellaan liitännäisiä, niiden hyötyjä ja rajoitteita sekä niiden merkitystä tämän työn toteutuksessa. Luvussa 4 määritellään liitännäistoteutuksen tarvitsemia toimintoja. Luvussa 5 tarkastellaan työn arkkitehtuurisuunnittelua. Luvussa 6 määritellään yksinkertainen korttipeli, jota käytetään sovelluksen toiminnan osoittamiseen. Luvussa 7 käydään läpi sovelluksen kehittämisen etenemistä, ongelmia ja ratkaisuja. Luvussa 8 esitetään yhteenveto työstä.

2. TAUSTAA

Tässä luvussa esitellään työn aihepiirin taustoja. Läpi käydään korttipelejä yleisesti, Magic: The Gathering -korttipelin sääntöjä yleisellä tasolla esimerkkinä korttipelistä ja korttipelien siirtämistä digitaaliseen muotoon.

2.1 Korttipeleistä yleisesti

Korttipelillä tämän työn yhteydessä tarkoitetaan niin kutsuttuja keräilykorttipelejä. Keräilykorttipelit ovat korttipelejä, joita pelataan erityisillä kutakin peliä varten kehitetyillä pelikorteilla. Painotuksesta ja markkinointitavasta riippuen nämä tunnetaan englanniksi termeillä Collectible Card Game (CCG), Trading Card Game (TCG) tai Living Card Game (LCG).

CCG ja TCG ovat käytännössä synonyymeja. Kuten nimistä voi päätellä, tämän tyypin korttipeleissä painotetaan korttien keräilyä ja vaihtamista suurena osana pakanrakennusta. Kortit luokitellaan harvinaisuuden mukaan eri ryhmiin niin, että harvinaisemmat kortit ovat ainakin teoriassa voimakkaampia kuin yleisemmät kortit. Harvinainen kortti voi myös vain olla parempi versio vähemmän harvinaisesta kortista. Tämä mahdollistaa tietyn “arkkityypin” pakkojen rakentamisen, vaikka pelaaja ei omistaisi parhaita mahdollisia arkkityypin käyttämiä kortteja. Toisaalta tämä myös tarkoittaa, että osa korteista on vain niin kutsuttuja täytekortteja. Toisaalta tämä myös johtaa innovaatioihin, kun pelaajat etsivät tapoja kiertää heikompien korttien huonoja puolia, tai parhaimmassa tapauksessa kääntävät niiden heikkouksia omaksi edukseen. Korttien määrän kasvaessa vanhojen ja uusien korttien yhdistäminen helposti johtaa korttien täysin uusiin käyttötapoihin.

LCG-tyypin korttipeleissä korttien kerääminen ja vaihtaminen on jätetty kokonaan pois. Kaikki pakkaukset sisältävät samat kortit, jolloin kaikilla pelaajilla on samat kortit käytettävissään. Tämän myötä LCG-korttipelit painottavat varsinaista pakanrakennusta korttien keräämisen sijaan. LCG-korttipelit ovat omalta osaltaan vastalause CCG- ja TCG-tyypin korttipeleille, joissa suurin rajoittava tekijä pakan suunnittelussa on yleensä käytettävissä oleva korttivalikoima. Toisaalta LCG-tyypin korttipeleissä rajoitetumpi korttivalikoima johtaa helposti kapeampaan valikoimaan toimivia arkkityyppejä, sillä pienemmällä korttivalikoimalla on epätodennäköisempää, että pelaajat löytävät arkkityyppejä, joita kehittäjät itse eivät tulleet ajatelleeksi.

Painotuksesta riippumatta korttipelit toimivat yleensä pohjimmiltaan samalla lailla: kukin pelaaja rakentaa käytössään olevista korteista oman pakan ja pelaa sillä yhtä tai useampaa pelaajaa vastaan. Suurin osa korttipeleistä on suunniteltu kaksinpeleiksi – tämän myötä usein osana korttipelien teemaa on kaksintaistelu (duel) – mutta useimmat tukevat joko suoraan tai fanien kehittämillä muutoksilla myös erilaisia moninpelimuotoja. Yleensä

muutokset ovat hyvin yksinkertaisia: Magicin “Two-Headed Ogre” -pelimuoto esimerkiksi muuttaa kaksintaistelun kahden hengen joukkueiden väliseksi mittelöksi, jossa koko joukkue pelaa oman vuoronsa yhtä aikaa.

Tavoitteena korttipeleissä on täyttää jokin pelin voittoehdoista. Voittoehtojen lisäksi useissa korttipeleissä on myös häviöehtoja. Tyypillisin sellainen on, että pelaaja häviää pelin, jos hänen pakastaan loppuvat kortit kesken. Yksinkertaisin voittoehto onkin, että kaikki muut pelaajat ovat pudonneet pois pelistä jonkin häviöehdon täyttymisen vuoksi.

Magicin tapauksessa oletusvoittoehto on pudottaa muut pelaajat pelistä jonkin häviöehdon täyttymisen kautta. Häviöehtoihin kuuluvat elinvoiman (life points) laskeminen nollille tai pakan loppuminen. Lisäksi peliin on vuosien mittaan lisätty paitsi kortteja jotka tuovat peliin lisää voitto- ja häviöehtoja, myös kortteja, jotka estävät joidenkin häviöehtojen täyttymisen.

2.2 Magic: The Gathering -säännöt pähkinänkuoressa

Magic: The Gathering oli ensimmäinen keräilykorttipeli (Kotha, 1998) ja on siksi asettanut genrelle vahvan pohjan. Suuri osa korttipeleistä lainaa ainakin osaa Magicin mekaniikoista. Tämän vuoksi sen sääntöjä voidaan hyödyntää korttipelimoottorin pohjan rakentamiseksi.

Teemaltaan Magic on velhojen taistelu: kaksi tai useampi velhoa – pelaajat – loitsii taikoja, joita pelin kortit edustavat. Kuten edellisessä kohdassa jo mainittiin, oletusvoittoehtona on tiputtaa muut pelaajat pelistä.

Magic on vuoropohjainen peli. Pelaajan vuoro on jaettu vaiheisiin (phase). Vaiheessa voi olla pakollisia askelia (step). Vuoron rakenne esitellään kuvassa 1. Vuoron vaiheiden sisältö on esitelty seuraavassa.

Kuva 1. *Magicin vuororakenne. Vuoro etenee vasemmalta oikealla. Yksittäisen vaiheen askeleet etenevät ylhäältä alas.*

Alkuvaihe (Beginning phase) sisältää askeleet, joissa pelaaja valmistautuu uuteen vuoroon. Pelaaja palauttaa kaikki korttinsa takaisin valmiuteen (Untap), suorittaa mahdolliset ylläpitotoimenpiteet (Upkeep) ja nostaa pakasta kortin (Draw). Poikkeuksena normaaliin vuoron toimintaan aloittaja ei nosta ensimmäisellä vuorollaan korttia, mikä tasapainottaa aloittamisesta saatavaa etua.

Päävaiheet (Main phase) 1 ja 2 ovat vaiheita, joissa pääasiallinen korttien pelaaminen tapahtuu: pelaaja voi pelata vuoronsa aikana yhden Land-kortin ja resurssiensa rajoissa kortteja, joita ei normaalisti voi pelata oman päävaiheen ulkopuolella. Tällaisia ovat yleensä esimerkiksi Sorcery- ja Creature-tyypin kortit.

Taisteluvaihe (Combat phase) on vaihe, jossa vuorossa oleva pelaaja voi halutessaan hyökätä Creature-korteillaan vastustajan kimppuun. Tässä vaiheessa pelataan taistelua edeltävät efektit (Beginning of Combat), valitaan hyökkääjät (Declare Attackers) ja puolustajat (Declare Blockers). Lopuksi lasketaan tapahtunut vahinko (Combat Damage) ja käsitellään taistelun jälkeiset tapahtumat (End of Combat).

Loppuvaiheessa (Ending phase) pelaajilla on aluksi viimeinen mahdollisuus pelata efektejä ja kortteja (End). Seuraavaksi vuoronsa lopettava pelaaja poistaa kädestään ylimääräiset kortit käsikorttirajaan asti, minkä jälkeen Creature-kortit palautuvat täyteen kuntoon ja “until end of turn” -efektit päättyvät (Cleanup).

Aina läpi käytävien toimenpiteiden lisäksi pelialueella olevilla korteilla voi olla sääntöjä, jotka laukaisevat (trigger) tietyn efektin tietyssä vaiheessa. Esimerkkeinä näistä ovat “at the beginning of your upkeep”, “at the end of the turn” ja “at beginning of combat”. Kyseiseen askeleeseen siirryttäessä kaikki tällaiset tapahtumat asetetaan pinoon. Vuorossa oleva pelaaja päättää efektien järjestyksen pinossa.

Yllä mainittujen tapahtumien lisäksi jokaisessa askeleessa Untap- ja Cleanup-askelia lukuun ottamatta pelaajilla on mahdollisuus käyttää efektejä ja pelata Instant-tyypin kortteja. Vuorossa olevalla pelaajalla on aina prioriteetti, eli jokaisen vaiheen aluksi ensin vuorossa olevalla pelaajalla on tilaisuus toimia. Jos pelaaja toimii, asetetaan toiminto pinoon käsittelyä varten. Jos pelaaja ei toimi, kiertää prioriteetti seuraavalle pelaajalle. Jos yhden prioriteettikierroksen aikana yksikään pelaaja ei toimi, siirrytään käsittelemään pinossa mahdollisesti olevia tapahtumia. Jokaisen tapahtuman käsittelyn jälkeen pelaajilla on jälleen tilaisuus toimia. Jos pinossa ei ole tapahtumia eikä kukaan toimi, askel loppuu ja siirrytään seuraavaan askeleeseen. Jos vaiheessa ei ole enää askelia, siirrytään seuraavaan vaiheeseen. Jos vaiheita ei ole enää jäljellä, siirtyy vuoro seuraavalle pelaajalle.

Vaiheiden lisäksi on muutama yleinen sääntö. Creature-tyypin korteilla on Summoning Sickness -sääntö. Säännön mukaan kyseisellä vuorolla pelialueelle tulleella kortilla ei voi hyökätä tai käyttää kykyjä, joiden hintaan kuuluu “tap”, eli kortin vinoon asettaminen aktivoinnin merkiksi. Sääntö koskee myös kortteja, jotka muuttuvat olennoiksi jonkin efektin myötä samalla vuorolla, kun ne tulevat pelialueelle.

Toinen yleinen sääntö koskee käsikorttien määrää. Normaalisti pelaajan enimmäiskäsikorttimäärä on seitsemän. Kortit saattavat vaikuttaa tähän määrään. Käsikorttien määrä tarkistetaan pelaajan oman vuoron loppuvaiheen Cleanup-askeleessa.

Tärkein Magicin sääntö on niin sanottu Kultainen Sääntö: jos pelin normaalit säännöt ja kortin sääntöteksti ovat ristiriidassa, kortin sääntöteksti on oikeassa. Yksinkertainen esimerkki tästä on avainsana Haste, jonka sisältävä kortti ei välitä Summoning Sickness -säännöstä. Viimeisenä yleisenä Magicin sääntönä on, että vuorossa voi pelata vain yhden Land-tyypin kortin, kuten yllä Päävaiheen kohdalla mainittiin.

2.3 Korttipelit ohjelmistoina

Teoriassa korttipelien kääntäminen ohjelmistoksi on helppoa. Korttipeleillä on pääasiassa hyvin selkeät ja yksinkertaiset perussäännöt. Suurin osa peleistä on jaettu vaiheisiin, joissa on selkeästi rajattu, mitä niissä tapahtuu, mitä niissä voi tehdä ja miten niissä edetään.

Ongelmana on, että kasvotusten pelatessa pelaajat yleensä ohittavat suuren osan vaiheista, ja jos toinen pelaaja hyppää liian pitkälle eteenpäin, on takaisin palaaminen yleensä suhteellisen helppoa. Myös inhimillisen virheen sattuessa pelaajat yleensä voivat sopia keskenään, miten tehty virhe korjataan. Yleensä se tapahtuu peruuttamalla yksi tai useampi vaihe taaksepäin. Varsinkin alkupuolella peliä on hyvin mahdollista, että yksi vuoro kestää vain muutaman sekunnin, sillä kaikkia vaiheita ei tarvitse käydä läpi.

Ohjelmiston puolella jokainen vaihe pitää käydä läpi oikeassa järjestyksessä ja virheitä ei saa tapahtua. Teoriassa tämä ei ole ongelma. Käytännössä ongelmaksi muodostuu ristiriita kahden pelaamisen kannalta oleellisen tekijän välillä. Toisella puolella on korttipelien strategian kannalta oleellinen osatekijä, salainen informaatio, toisella pelaamisen mukavuuden kannalta tärkeä pelin sujuvuus.

Seuraava esimerkki ongelmasta on Magicista. Matti pelaa suoraviivaista pakkaa, joka hiukan kärjistäen toimii vain oman vuoronsa päävaiheessa (Main phase) ja taisteluvaiheessa (Combat phase). Mikko pelaa pakkaa, jonka tarkoitus on voittaa pitkittämällä peliä ja keräämällä resursseja, kunnes voi turvallisesti pelata kuvaannollisen ässän hihastaan ja voittaa sen avulla.

Kasvotusten peli etenee seuraavasti. Matti pelaa kortin. Mikko arvioi nopeasti, onko kortista riittävästi uhkaa, että se pitäisi pysäyttää. Jos ei, peli etenee normaalisti. Jos kortti on liian uhkaava, Mikko ilmoittaa reagoivansa korttiin ja yrittää pysäyttää sen omalla kortillaan, johon saattaa liittyä ehdollista toimintaa, johon Mikon pitää vastata.

Ohjelmiston puolella tilanne on monimutkaisempi. Magicissa jokaisella pelaajalla on oikeus reagoida jokaiseen tapahtumaan. Matin pelatessa kortin ohjelmiston pitäisi ensin tarkistaa Mikolta, reagoiko hän Matin pelaamaan korttiin. Tämän jälkeen myös Matilla

on tilaisuus reagoida. Vasta jokaisen pelaajan päätettyä olla reagoimatta korttiin varsinainen tapahtuma prosessoidaan. Jos Mikko reagoi, on tämä uusi tapahtuma ja jälleen kaikilta pelaajilta täytyy tarkistaa reaktio. Ongelmallista tässä on riittävän varmistuksen ja pelin etenemisen tasapaino. Jatkuva “en reagoi” -palautteen antaminen hidastaa peliä huomattavasti ja voidaan kokea turhauttavaksi, varsinkin jos pelaaja on tottunut kasvotusten pelattaessa viestittämään informaation minimaalisella eleellä tai tarkistamaan reaktion vastustajalta vain tilanteessa, jossa sillä on suuri merkitys pelin etenemisen kannalta ja on oletettavaa, että vastustaja pystyy reagoimaan.

Yksi tapa – joka toimii yleensä moitteetta pelattaessa tekoälyä vastaan – on antaa pelaajan etukäteen ilmoittaa, missä kohdissa peliä hän haluaa varmistuksen reagoinnistaan. Tämä ei kuitenkaan ole täydellinen ratkaisu: jokainen pysäytys on vastustajalle lisäinformaatiota, joka kertoo, että vastapuolella on kortteja, joita voi käyttää tässä tilanteessa. Magicissa vastustajan pakan pelaamat värit ja tähän mennessä nähdyt kortit – ja mahdolliset aiemmat ottelut – auttavat pelaajaa rajaamaan vaihtoehdot tiettyihin kategorioihin, parhaimmillaan yksittäiseen korttiin. Ainoa tapa estää tämän informaation antaminen on pysäyttää kaikissa kohdissa, jolloin palataan alkuperäiseen ongelmaan. Toinen ongelma on, että pysäytyksen ollessa mahdollinen vain ennalta määrätyissä kohdissa saatetaan pelaaja pakottaa varaamaan pysäytys yksittäiseen vaiheeseen peliä, vaikka pysäytystä tarvitsisi vain kerran koko pelin aikana.

Toinen ratkaisu on käyttää kiinteää odotusaikaa, jonka aikana pelaajat voivat käyttää keskeytä-komentoa ja “varata” vuoron pelata kortteja. Tämän ongelmana on, että joko jokaiselle pelaajalle annetaan vuorollaan aikaa reagoida – mikä lisää odotusten kestoa ennestään – tai joudutaan tekemään kompromissi esimerkiksi Magicin sääntöjen ja pelin nopeuden välillä: korttien pelaamisen järjestyksellä on joskus hyvin suuri merkitys. Toinen ongelma on, että jokaisen vaiheen – myös muuten yleensä täysin mekaanisen – on kestettävä vähintään valitun odotusajan verran. Odotusajan täytyy myös esimerkiksi latenssin huomioimiseksi ja reagoimisen mahdollistamiseksi olla useamman sekunnin mittainen, mikä voi hidastaa useista valinnaisista vaiheista koostuvaa peliä huomattavasti.

2.4 Olemassa olevat korttipelimoottorit

Ennen oman korttipelimoottorin suunnittelua on hyvä suorittaa katsaus olemassa oleviin vastaaviin ohjelmiin. Tässä työssä korttipelimoottorin määritelmänä käytetään seuraavaa: *korttipelimoottori on toteutusalustasta riippumatta sovellus, jonka pääasiallinen tarkoitus on mahdollistaa yhden tai useamman olemassa olevan korttipelin pelaamisen digitaalisesti ilman fyysisiä kortteja*. Määrittely sulkee tarkoituksellisesti pois sovellukset, jotka tukevat ainoastaan korttipelejä, joilla ei ole fyysistä vastinetta. Tällaiset korttipelit on suunniteltu alusta alkaen digitaalisiksi, eivätkä ne yleensä kärsi ongelmista, joita fyysisen korttipelin siirtäminen digitaaliseen ympäristöön aiheuttaa. Varsinaisen korttipelin

pelaamisen lisäksi korttipelimoottori saattaa tarjota muita toimintoja, kuten pakanrakennusta, korttikokoelman hallintaa, korttien vaihtamista ja muita toimintoja, joiden avulla simuloidaan korttipelikokemusta kokonaisuutena.

Korttipelimoottoreista ehkä tunnetuin on Magic: The Gathering Online (Wizards of the Coast, 2002), joka nimensä mukaan on digitaalinen vastine korttipelille Magic: The Gathering. Magic Online vaatii käyttäjätilin, jolla kirjaudutaan sovelluksen palvelimelle käyttäen kehittäjän sivuilta ladattavaa asiakassovellusta. Käyttäjätilin luominen on maksullista, mutta hintaan sisältyy aloituspaketti. Itse ohjelman käyttäminen on ilmaista. Lisäkortit täytyy erikseen ostaa.

Magic Online edustaa yhtä korttipelimoottorien ääripäätä. Se sisältää tuen pakan rakentamiselle, sovellustuen sääntöjen mukaiselle pelaamiselle, tuen eri pelimoodeille korttirajoitus tarkasteluineen ja korttien vaihtelun. Nimensä mukaan Magic Online kuitenkin tukee vain Magic: The Gathering -peliä.

Magic Onlinen lisäksi Wizards of the Coast on julkaissut myös pienempiä korttipelimoottoreita, joissa on otos keskenään tasapainotettuja pakkoja, joita joko ei voi muokata ollenkaan tai voi muokata vain hyvin rajoitetusti. Viimeisin esimerkki näistä on Magic: The Gathering - Duels of Planeswalkers 2015 (Wizards of the Coast, 2014), joka on julkaistu sekä konsoleille että PC:lle. Tätä ei kuitenkaan tule sekoittaa samannimiseen vuonna 1997 julkaistuun versioon, joka tunnetaan myös pelin fantasiamaailman nimellä, Shandalar (Wizards of the Coast, 1997). Toisin kuin uudemmat versiot, Shandalar sisälsi pakkaeditorin ja siedettävän tekoälyn, joka kykeni vaihtelevalla menestyksellä pelaamaan myös pelaajan tekemillä pakoilla. Tämän lisäksi Shandalar sisälsi myös kampanjamoodin, jossa pelaaja hiljalleen keräsi uusia kortteja voittamalla otteluita kartalla kohtaamiltaan vastustajilta. Tavoitteena Shandalarin kampanjassa oli valloittaa jokaista magian väriä vastaava linnoitus maailmankartalta.

Wagic: The Homebrew (Wagic) on yhden miehen Playstation Portable -projektista alkunsa saanut Magic: The Gathering -fanisovellus, joka tarjoaa toimivan pakanrakennuksen lisäksi pelin sääntöjen toteutuksen ja tekoälyn, jota vastaan pelata. Wagicia on kehitetty pääasiassa Playstation Portablelle ja älypuhelimille, mutta se tarjoaa tuen myös Windowsille ja Linuxille. Avoimen kehitystavan myötä sovellus tukee hyvin myös modausta. Wagicia onkin modattu tukemaan myös muita, Magicin kaltaisia korttipelejä.

Wagic: The Homebrew on esimerkki korttipelimoottorista, joka on saanut inspiraationsa ja – Wagicin tapauksessa alun perin myös pääasiallisen sisältönsä – Magicista. Sovellus on tämän myötä alun perin kehitetty pääasiassa Magicin pelaamista varten. Oma ongelmansa on, että korttien grafiikat ja kaikki Magicin symbolit kuuluvat Wizards of the Coastille. Kehittäjät eivät tämän vuoksi voi itse levittää tätä sovellukselle oleellista graafista sisältöä, vaan näiden etsiminen jää käyttäjien ongelmaksi.

Wagicin kanssa samaan luokkaan kuuluvat myös MTGPlay (Pesonen, 2000) ja Magic Workstation (Magi-Soft Development, 2002). Molemmat ovat saaneet inspiraationsa Magicista ja molemmat on myös suunniteltu täysin Magicia ajatellen. Erona Wagiciin sekä MTGPlay että Magic Workstation ovat jättäneet tuen pelin säännöille kokonaan pois. Varsinainen pelaaminen ja sääntötulkinnat jäävät siis täysin sovelluksen käyttäjien vastuulle.

Vaikka molemmat sovellukset on suunniteltu pääasiallisesti Magicin pelaamiseen, voi niillä käytännössä pelata myös muita korttipelejä, sillä sovellusten tarjoamat perustoiminnot ovat hyvin yleisiä. MTGPlay ja Magic Workstation sisältävät tuen pakan rakentamiselle ja verkon ylitse pelaamiselle. Itse korttien kuvat sekä pakan rakentamisen hakutoiminnon vaatimat korttien tiedot täytyy ladata sovelluksiin erikseen. Varsinainen pelaaminen vaatii, että pelaajat tuntevat pelattavan pelin hyvin, sillä kommunikaatio pelaajien välillä sovelluksen osalta on rajoitettu pelaajan kontrollissa olevien korttien liikutteluun peli-ikkunassa, eri korttien highlight-toimintoon ja sisäänrakennettuun yksinkertaiseen chat-toimintoon. Ilman erillisen VoIP-ohjelman käyttöä pelaaminen tuntemattoman kanssa voi olla hyvin työlästä. Toisaalta VoIP-keskustelua hyödynnettäessä nämä sovellukset pääsevät tietyssä mielessä lähimmäksi kasvotusten pelaamisen simulointia niin hyvine kuin huonoine puolineen.

Korttipelien suhteellisen hyvän menestyksen vuoksi ei ole yllättävää, että niistä on tehty myös useita adaptaatioita paitsi PC:lle, myös kannettaville konsoleille. Alustasta johtuen tämän tyyppiset korttipelimoottorit poikkeavat jossain määrin moderneista PC-vastineistaan. Näistä erityisen maininnan ansaitsee Yu Gi Oh! -korttipeli (Konami, 1999). Pääasiassa japanissa tunnetusta Yu Gi Oh!:sta on tehty kannettaville konsoleille (Gameboy Advance, Nintendo DS) jo vuodesta 2004 asti uusi vuosittainen World Championship -versio. Nämä versiot paitsi sisältävät suuren otoksen julkaistuista korteista pelaajan kerättäväksi, myös noudattavat kyseisen vuoden kilpailurajoituksia eri korteille. Valmiiden tekoälyvastustajien lisäksi kannettavat konsolit tukevat myös linkkikaapelin kautta muita vastaan pelaamista ja tietenkin korttien vaihtamista. Pelit myös pääasiassa simuloivat oikean korttipelin keräilytapaa. Niissä ostetaan lisäkorttipakkauksia (booster pack) korttikokoelman kasvattamiseksi käyttämällä pelin omaa virtuaalirahaa. Virtuaalirahaa pelaaja kerää voittamalla otteluita tekoälyvastustajia vastaan. Tämän lisäksi osassa peleistä on myös muita, itse korttipeliin vain epäsuorasti liittyviä pelillisiä ominaisuuksia, jotka tuovat pelin yksinpeliin lisämakua.

3. LIITÄNNÄINEN TOTEUTUSTEKNIIKKANA

Vaikka korttipeleillä on useita jaettuja ominaisuuksia, niiden ominaisuudet ja mekaniikat voivat olla hyvin erilaisia. Geneerisen toteutuksen mahdollistamiseksi suuri osa sovelluksen toiminnasta täytyy muuttaa sopimaan kullekin korttipelille erikseen. Liitännäisten käyttö on tässä työssä valittu ratkaisuksi tämän muunneltavuuden toteutukseen.

Tässä luvussa tarkastellaan liitännäisiä yleisesti, niiden käytön sovelluksia, etuja sekä miten niitä hyödynnetään tämän työn puitteissa.

3.1 Liitännäiset yleisesti

Liitännäinen (plugin) on termi, jonka rajat eivät ole kovin tarkkaan määriteltyjä. Määritelmät yleensä – kuten Merriam-Websterin (2014) "*a small piece of software that adds a feature to a larger program or makes a program work better*" – painottavat liitännäisen ja pääohjelman kokoeroa: tällaisten määritelmien mukaan liitännäinen on vain pieni, valinnainen komponentti, joka tuo jonkin pienehkön lisäominaisuuden tai toiminnallisuuden suurempaan pääohjelmaan. Tämä ei kuitenkaan ole ainoa tapa hyödyntää liitännäisiä.

Mayer (Mayer, 2003) tuo esiin näkökulman, jonka mukaan on mahdollista luoda sovelluksia, jotka koostuvat pääasiassa liitännäisistä, ja esittelee suunnittelumallin, jossa pääohjelma on suhteessa pieni ja pelkistetty, mutta laajennettavissa liitännäisten avulla. Dietrich (Dietrich, 2007) myötäilee liitännäisten käytön etuja ja luokittelee lisäksi liitännäismallit kahteen sukupolveen: ensimmäisen sukupolven liitännäisiin, jotka tuovat vain hyvin rajoitettua lisätoiminnallisuutta olemassa olevaan sovellukseen ja toisen sukupolven liitännäisiin, jotka tarjoavat tuen myös omille lisäpalikoilleen. Esimerkkeinä toisen sukupolven mallia noudattavista sovelluksista Dietrich mainitsee Eclipsen ja Java Plugin Frameworkin (Dietrich, 2007).

Määritelmästä riippumatta liitännäisen käyttö pitää ottaa huomioon jo sovelluksen kehitysvaiheessa. Käytännössä tämä tapahtuu toteuttamalla sovellukselle yksi tai useampi rajapinta (interface), jonka kautta liitännäinen ja sovellus vaikuttavat toisiinsa. Rajapinta on ainoa yhteys pääsovelluksen ja liitännäisen välillä. Hyvin suunniteltu rajapinta antaa laajat työkalut liitännäiskehittäjien käyttöön ilman, että heidän täytyy tietää pääsovelluksen sisäisestä toiminnasta mitään.

Tämän työn puitteissa liitännäisen määrittelyksi annetaan seuraava: *Liitännäinen on toista sovellusta varten kehitetty apusovellus, jonka pääsovellus voi käynnistää dynaamisesti ja joka antaa käynnistävälle sovellukselle lisää toiminnallisuutta sen tarjoamaa liitännäisrajapintaa hyödyntäen*. Liitännäinen on siis käytännössä ajonaikainen lisäosa, joka voidaan tarvittaessa käynnistää jonkin ominaisuuden käyttöön ottamiseksi. Kuten

Merriam-Websterin määritelmästä voi päätellä, perinteisesti nämä ominaisuudet ovat pieniä suhteessa pääohjelmaan, esimerkiksi uuden tiedostotyypin käsittelyyn vaadittava toiminnallisuus. Yllä esitetty määritelmä jättää tarkoituksellisesti sovellusten kokojen vertaamisen pois.

3.2 Liitännäispohjainen sovelluskehitys

Tämän työn kannalta merkittävä käyttötapa liitännäisten hyödyntämiselle on Mayerin esittämä liitännäispohjainen sovelluskehitys ja tähän tarkoitukseen suunniteltu liitännäismalli. Liitännäispohjaisessa sovelluskehityksessä lähtökohtana on, että pääohjelma on mahdollisimman kevyt nopean käynnistämisen ja minimaalisen muistin käytön mahdollistamiseksi. Koska kaikkia – sovelluksen koosta riippuen mahdollisesti edes huomattavaa osaa – ominaisuuksia ei tarvita jokaisen käyttökerran yhteydessä, tämä vähentää huomattavasti sovelluksen vaatimia resursseja normaalikäytössä verrattuna sovellukseen, joka lataa kaikki ominaisuutensa joka käynnistyskerralla. Sovellus myös käynnistyy nopeammin, kun vain murto-osa kaikesta toiminnallisuudesta ladataan käynnistyksen yhteydessä. (Mayer, 2003)

Mayerin mallissa poikkeuksena perinteiseen liitännäiskäytäntöön mahdollisimman suuri osa sovelluksen toiminnallisuudesta toteutetaan liitännäisillä, jotka ladataan dynaamisesti vasta, kun käyttäjä niitä tarvitsee. Koska suurin osa toiminnallisuudesta on liitännäisten puolella, sovelluksen osia voidaan tarpeen mukaan korvata ja päivittää ilman, että sovelluksen varsinaista ydintä joudutaan muokkaamaan. Tämä helpottaa ainakin teoriassa sovelluksen ylläpidettävyyttä huomattavasti. Mayerin mukaan mallin avulla voidaan helpottaa suurten järjestelmien monimutkaisuutta pilkkomalla niitä pienempiin, helpommin käsiteltäviin moduuleihin. Mallin avulla voidaan myös vastata sovelluksen ajamisen aikana ilmeneviin tarpeisiin. Hyvänä esimerkkinä tästä on palvelinsovellus, jota syystä tai toisesta ei voida tai haluta käynnistää uudelleen, mutta joka voi dynaamisesti käynnistää myös sellaisen liitännäisen, joka on kehitetty palvelimen käynnistämisen jälkeen. (Mayer, 2003)

Kritiikkinä Mayerin malliin voidaan huomauttaa, että vaikka liitännäisosuudet voidaan tarvittaessa korvata uusilla, mikä osaltaan helpottaa ylläpidettävyyttä, samaa ei voida sanoa ohjelman ydinkoodista. Kaikki muutokset ydinkoodiin voivat potentiaalisesti rikkoa aiempien liitännäisten toiminnan, mikä nostaa huomattavasti kynnystä tehdä edes tarpeellisiksi koettuja muutoksia ydinkoodiin. Tätä voidaan kiertää kirjoittamalla vanhojen toimintojen rinnalle uusia ja suosittelemalla niiden käyttöä tulevaisuudessa, mutta tämä sotii suoraan suunnittelumallin periaatteita vastaan. Tämä asettaa ohjelman ydinkoodille huomattavasti korkeammat laatuvaatimukset normaaliin sovelluskehitykseen verrattuna. Ongelmaa vaikeuttaa entisestään se, että ydinkoodin kehittäjä ei voi mitenkään ennustaa, mihin kaikkeen ydinohjelmaa ja sen rajapintaa pyritään tulevaisuudessa venyttämään.

Mayerin malli ei ole ainoa näkemys liitännäispohjaisesta sovelluskehityksestä. Tämän työn puitteissa se toimii hyvänä esimerkkinä tämän tyyppisestä sovelluskehityksestä. Työn puitteissa liitännäispohjaisella sovelluskehityksellä viitataan kuitenkin kaikkeen sovelluskehitykseen, jossa liitännäiset ovat merkittävässä roolissa riippumatta siitä, missä määrin ne noudattavat Mayerin mallin linjaa.

3.3 Liitännäissovellusten hyödyt ja rajoitteet

Liitännäisten hyötyjä ja rajoitteita täytyy tämän työn puitteissa tarkastella kahdesta näkökulmasta: "kevyen" tai "perinteisen" liitännäisen näkökannalta, jossa pääsovellus sisältää pääosan toiminnallisuudesta, johon liitännäinen tuo pieniä lisäominaisuuksia, kuten tuen uudelle tiedostotyypille tai graafisen ilmeen muutoksia, ja suurempia liitännäismoduuleja vaativan liitännäispohjaisen sovelluskehityksen kannalta, jossa sovelluksen varsinainen sisältö on liitännäisissä.

Pääsovelluksen käyttäjän näkökulmasta perinteisten liitännäisten ilmeisin hyöty on niiden suoma räätälöinnin mahdollisuus. Kattavat liitännäisrajapinnat sallivat hyvin monipuolisen kirjon lisäominaisuuksia, joita loppukäyttäjät voivat valita ja lisätä käyttämäänsä sovellukseen mielensä mukaan. Tarpeen vaatiessa osaava käyttäjä voi myös itse kehittää tarvitsemiaan lisäominaisuuksia sovellukseen.

Kehittäjän näkökulmasta perinteinen liitännäinen tarjoaa joustavuutta sovellukselle ja sallii tiettyjen osien kehittämisen delegoinnin käyttäjille. Käytännössä tämä tarkoittaa, että kehittäjä voi ulkoistaa esimerkiksi sovelluksen erilaisten ulkoasujen kehittämisen käyttäjille ja ottaa tulevaisuudessa kehitettävät tiedostomuodot huomioon tarjoamalla tuen niitä tulkkaaville liitännäisille.

Liitännäispohjaisen sovelluskehityksen puolella sekä Mayer (Mayer, 2003) että Wagner (Wagner, 2007) listaavat etuihin kompleksisten järjestelmien yksinkertaistamisen. Hyvinkin monimutkaiset järjestelmät voidaan pilkkoa pieniin, itsenäisiin moduuleihin. Monimutkaisen kokonaisuuden jakaminen pienempiin kokonaisuuksiin laskee yksittäisen osan kokoa ja sitä myötä monimutkaisuutta merkittävästi. Tämän seurauksena yksittäistä osaa on helpompi käsitellä, suunnitella, ohjelmoida ja testata. Yksittäisen osan monimutkaisuuden vähenemisen lisäksi kehitystä helpottaa, ettei kehittäjän tarvitse tietää muista ohjelman osista enempää, kuin mitä omalle osalle tarjotut ja siltä vaaditut rajapinnat hänelle kertovat.

Wagner kuitenkin huomauttaa, että sovelluksen pilkkominen pienempiin osiin ei suoranaisesti poista monimutkaisuutta: monimutkaisuus vain siirtyy ohjelmointitasolta mallinnustasolle. Tämän lisäksi esille nousee myös moduulien yhteensovittamisen myötä syntyvän "liimakoodin" mukanaan tuomat ongelmat. (Wagner, 2007)

Liitännäiset mahdollistavat myös sovelluksen jatkokehittämisen sekä kehittäjien että käyttäjien toimesta ilman, että sovelluksen ydinkoodiin tarvitsee koskea. Perinteiset liitännäiset sallivat uusien ominaisuuksien lisäämisen ja sovelluksen laajentamisen rajapinnan puitteissa. Liitännäispohjaisten sovellusten myötä tämä menee vielä pidemmälle: koska suurin osa toiminnallisuudesta sijoittuu liitännäisiin, voidaan sovelluksen osia kehittää ja korvata vapaasti sovelluksen vaatimusten mukaan. Wagner huomauttaa, että tämä ei rajoitu pelkästään valmiin sovelluksen jatkokehitykseen: muuttuvat vaatimukset ovat sovelluskehityksessä arkipäivää (Wagner, 2007). Uusien ominaisuuksien lisäämisen helppous auttaa uusien ominaisuuksien nopeaa kehittämistä sovellukseen.

Liitännäinen tarjoaa paljon mahdollisuuksia, mutta toisaalta se on luonnostaan myös rajoittunut. Käytännössä liitännäisen mahdollisuudet rajautuvat liitännäisrajapinnan puitteisiin. Kaikki informaatio pääsovelluksen ja liitännäisten välillä kulkee rajapinnan kautta. Toinen potentiaalinen ongelma on tiedon välittäminen liitännäiseltä toiselle pääohjelman kautta, mikä tapahtuu tarjottujen rajapintojen ehdoilla.

3.4 Liitännäisten käyttösovellukset

Liitännäisten perinteinen käyttötarkoitus on ollut sallia kolmannen osapuolen kehittäjien itsenäisesti lisätä olemassa olevaan sovellukseen uutta toiminnallisuutta. Yleinen tapa hyödyntää liitännäisiä on yksinkertaisen toiminnon lisääminen selaimeen. Hyvä esimerkki tästä on Adobe Flash Player (Adobe Systems, 1996). Tämän työn kannalta kiinnostavampia ovat kuitenkin liitännäispohjaisen sovelluskehityksen esimerkit, joista enemmän seuraavaksi.

Laajalti tunnettuna esimerkkinä liitännäisten hyödyntämisestä sovelluskehityksessä ja liitännäispohjaisesta sovelluskehityksestä voidaan antaa Eclipse. Eclipse on avoimen lähdekoodin ohjelmankehitysympäristö, jonka kehitystä ajaa Eclipse Foundation (2004). Eclipse tarjoaa itsessään vain kehitystyöhön tarvittavan rungon, jonka päälle käyttäjä voi valikoida alati kasvavasta liitännäisten kirjosta tarpeitaan vastaavia lisäpalikoita. Korkeammalla tasolla kyseessä voi olla tuki tietylle ohjelmointikielelle tai sovellusalustalle, alemmalla tasolla esimerkiksi Android-kehitykseen (Google, 2008) tarkoitettuja kohdealuekohtaisia apuvälineitä virheiden paikallistamiseen. Myös tämän työn osana kehitettävän korttipelimoottorin kehitystyö tapahtuu Eclipsellä.

Esimerkkinä tämän työn tavoitteen kaltaisesta sovelluksesta annettakoon Himmelspachin työssään esittelemä James II (Himmelspach, 2007). Sovelluksen tavoitteena on tarjota geneerinen simulaatioympäristö, jonka avulla voidaan suorittaa hyvin monipuolisia simulaatioita hyödyntämällä liitännäispohjaista sovelluskehitystä. James II tarjoaa rungon, jonka päälle valitaan tarvittavat liitännäiset halutun simulaatioskenaarion luomiseksi. Yhtenäinen runko sallii samojen simulaatioiden toistamisen eri tahojen toimesta sekä tekee tuloksista paremmin vertailtavia. Sovellus myös nopeuttaa simulaatioiden laatimista, sillä

käyttäjien tarvitsee vain valita simulaatioonsa tarvitsemansa moduulit täysin uuden simulaattorin kehittämisen sijaan.

3.5 Liitännäiset geneerisessä korttipelimoottorissa

Edellä on tarkasteltu liitännäisiä yleisesti, liitännäispohjaista sovelluskehitystä Mayerin mallia esimerkkinä käyttäen, liitännäisten hyötyjä ja haittoja sekä esimerkkejä liitännäisten käytöstä. Seuraavaksi tarkastellaan liitännäisten käyttöä tämän työn puitteissa.

Tämän työn puitteissa toteutettavan geneerisen korttipelimoottorin tarkoituksena on tarjota pelin säännöt toteuttava tuki periaatteessa mille tahansa korttipelille. Käytännössä tavoite on hyvin epärealistinen toteutettavaksi normaalilla tavalla: korttipelit ovat monimuotoisia ja käytännössä jokainen vaatisi oman toteutuksensa. Tämän vuoksi työssä turvaudutaan Mayerin mallia mukailevasti liitännäispohjaiseen suunnitteluun: tavoitteena on luoda geneerinen runko sovellukselle ja jättää varsinainen korttipelikohtainen tulkitseminen liitännäismoduulien toteutettavaksi. Geneerisen rungon vastuulle jää täten yksinkertaisten, geneeristen toimintojen suorittaminen ja rajapintojen tarjoaminen liitännäisille erilaisten korttipelien toteutusta varten.

Sovelluksen toteutuksen kannalta suurin haaste on rajata korttipelimoottorin geneerinen osuus oikein. Sovelluksen täytyy tukea tarpeeksi kattavasti erilaisia toimintoja menemättä kuitenkaan liian syvälle yksityiskohtiin.

4. LIITÄNNÄISTEN TUNNISTAMINEN

Korttipelimoottorin toteutuksen kannalta on oleellista paitsi tunnistaa korttipelien tarvitsemat geneeriset ominaisuudet, myös osata jakaa ne liitännäistoteutuksen kannalta järkevästi sopiviin kokonaisuuksiin. Siksi tässä luvussa määritellään liitännäispohjaisen toteutuksen kannalta oleellisia ominaisuuksia. Näihin kuuluvat vyöhykkeet, kortit, korttien vaatimat toiminnot ja korttipelin vuororakenne. Jako perustuu lähtökohtaan, että kortit ovat korttipelissä keskeisessä osassa. Tämän myötä kortit itsessään ja niiden vaatimat toiminnot voidaan nähdä olennaisina kokonaisuuksina. Vyöhykkeet sisältävät kortteja ja kortit siirtyvät niiden välillä, joten myös vyöhykkeet on järkevä rajata omaksi kokonaisuudekseen. Vuororakenne ei suoraan liity kortteihin, mutta koska se määrää itse korttipelin etenemisen, myös vuororakenne on oleellinen kokonaisuus.

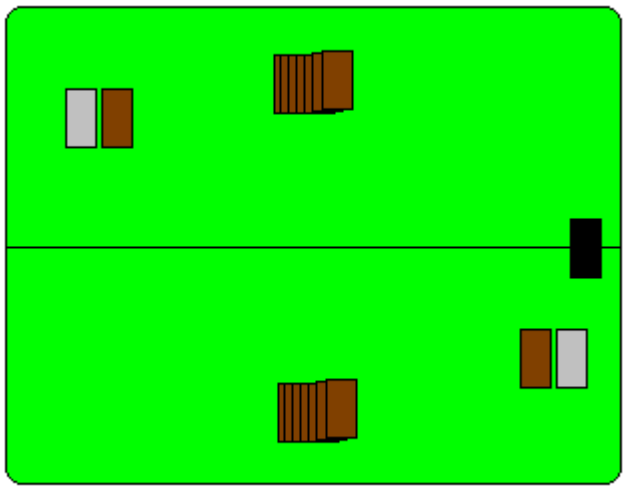
4.1 Vyöhykkeet

Pelipöydän ja varsinaisen pelaamisen toimintojen suunnittelu jakautuu kahteen osaan. Ensin täytyy määritellä vaadittavat vyöhykkeet (Zones). Tämän jälkeen määritellään kortteihin ja vyöhykkeisiin vaikuttavat mekaniikat. Vyöhykkeellä tarkoitetaan tässä työssä korttipelin sääntöjen määrittelemää aluetta, joka voi sisältää kortteja.

Vyöhykkeiden etsimisessä käytetään esimerkkinä Magic: The Gathering -korttipeliä. Se sisältää seuraavat vyöhykkeet: pakka (Library), käsikortit (Hand), pelialue (Battleground) poistopakka (Graveyard) ja poistoalue (Exile). Näiden lisäksi myös pino (Stack) voidaan nähdä vyöhykkeenä. Kuvassa 2 on esimerkki Magicin pelipöydästä.

Magicin vyöhykkeistä geneerisiä ovat pakka, ja käsikortit ja poistopakka. Nämä kuuluvat lähes poikkeuksetta kaikkiin korttipeleihin, ja yleisesti ottaen ne eivät merkittävästi eroa pelien välillä. Hyvänä esimerkkinä geneerisyydestä voidaan mainita, että myös moni perinteinen korttipeli sisältää nämä kolme vyöhykettä – pakka ja poistopakka yleensä vain ovat yhteiset kaikille pelaajille.

Pakka sisältää pelin alussa lähes poikkeuksetta kaikki kyseisen pelaajan yksittäisessä pelissä käytössä olevat kortit. Pakan ominaisuuksiin kuuluu, että korttien järjestyksellä on väliä ja kortit ovat kuvapuoli alaspäin (face-down). Jokaisella pelaajalla on oma pakka. Yleisiä pakan toimintoja ovat sekoitus (shuffle) ja korttien siirtyminen toiselle vyöhykkeelle – yleensä käsikortteihin. Pelitapahtumana harvinaisempia mutta useammassa pelissä esiintyviä toimintoja ovat kortin siirtyminen toiselta vyöhykkeeltä joko pakan päälle tai alle ja päällimmäisen kortin paljastaminen molemmille pelaajille joko hetkellisesti tai toistaiseksi.



Kuva 2. *Esimerkki Magicin pelipöydästä. Pelialue (vihreä) on jaettu selvyyden vuoksi pelaajien kesken kahtia. Oikealla alhaalla näkyy alemman pelaajan pakka (ruskea) ja poistopakka (harmaa). Yhteinen poistoalue (musta) sijaitsee oikeassa reunassa. Keskellä on kullakin puolella pelaajien käsikortit.*

Käsikortit-vyöhyke koostuu korteista, joita pelaaja on nostanut (draw) pakasta mutta ei ole vielä pelannut. Käsikortit ovat näkyvillä vain vyöhykkeen omistajalle (owner), ellei jokin toiminto anna tilapäisesti näkyvyyttä yhdelle tai useammalle vastustajalle. Käsikorttien ominaisuuksiin kuuluu, että ne voidaan pelata (play), yleensä tiettyä resurssikustannusta (mana cost) vastaan, jolloin ne siirtyvät pinoon. Jokaisella pelaajalla on omat käsikortit. Yleinen käsikortteihin liittyvä toiminto niiden pelaamisen ohella on yksittäisen kortin siirtäminen toiselle vyöhykkeelle, yleensä poistopakkaan. Toinen yleinen käsikortteihin liittyvä toiminto on yksittäisen kortin satunnainen valinta, esimerkiksi poistopakkaan siirtämistä varten.

Poistopakka on vyöhyke, johon kortit siirtyvät, kun ne on käytetty tai kun ne poistuvat pelialueelta. Poistopakan kortit ovat kaikille pelaajille näkyviä. Poistopakan ominaisuuksiin kuuluu, että sen järjestyksellä on osassa korttipeleistä merkitystä, osassa ei. Jokaisella pelaajalla on yleensä oma poistopakka.

Pelialue on korttipeleissä vaihtelevin vyöhyke. Magicissa se on jaettu tai neutraali alue, jossa jokaisella kortilla on sekä omistaja (owner) että kontrolloija (controller). Kortin omistaja on se pelaaja, jonka pakkaan kortti kuuluu. Kortin kontrolloija on se pelaaja, jonka hallussa kortti sillä hetkellä on. Omistaja on staattinen, kontrolloija voi vaihtua pelin edetessä. Vaikka pelialue on Magicissa jaettu, joissakin korttipeleissä pelialue on jaettu pienempiin “alivyöhykkeisiin”, jotka kuuluvat pelaajille samalla tavalla kuin pelaajan pakka tai poistopakka.

Pino on vyöhyke, joka toimii LIFO-periaatteen mukaisesti seuraavalla tavalla. Kun pelaaja pelaa kortin, se menee ensin pinoon. Tämän jälkeen kaikilla pelaajilla on mahdollisuus joko pelata kortti pinon päälle tai jättää reagoimatta (pass). Pinoa kasvatetaan, kunnes kaikki pelaajat ovat jättäneet reagoimatta yhden täyden kierroksen ajan. Seuraavaksi pinoon asetetut kortit käsitellään (resolve) yksi kerrallaan. Jokaisen käsitellyn kortin jälkeen jokaisella pelaajalla on jälleen tilaisuus reagoida ennen seuraavan kortin käsittelyä. Kaikki korttipelit eivät käytä pinoa samalla tavalla tai ollenkaan, mutta käytännössä pinon avulla voidaan abstrahoida suurin osa vaihtoehtoisista korttien käsittelytavoista.

Poistoalue on Magicin uusin nimetty vyöhyke. Se on jaettu vyöhyke, jonne pelistä poistetut (Exiled) kortit siirretään. Kortteja siirretään poistoalueelle joko pysyvästi tai väliaikaisesti. Pysyvästi siirrettyihin kortteihin ei voi kohdistaa (target) kykyjä. Väliaikaisesti poistetut kortit palaavat ne poistaneen kortin sanelemin ehdoin, yleensä kyseisen vuoron lopuksi. Poistoalueen kortit voivat olla joka kuvapuoli ylös- tai alaspäin riippuen siitä, miten ne ovat poistoalueelle päätyneet. Pelimekaanisesti poistoalue on jaettu vyöhyke, mutta selvyyden vuoksi pelaajat yleensä pitävät poistoalueelle siirretyt kortit esimerkiksi poistopakkansa vieressä.

Magicin vyöhykkeistä saadaan johdettua vyöhykkeen perusominaisuudet: vyöhykkeellä voi olla omistaja, mutta se voi myös olla neutraali tai jaettu. Vyöhykkeen korttien kuvapuolet voivat olla näkyvissä joko kaikille pelaajille tai vain yhdelle pelaajalle, tai vyöhykkeen korttien kuvapuolet voivat olla alaspäin kaikille. Vyöhykkeellä olevien korttien näkyvyyttä – joko kaikkien, tai osan niistä – täytyy myös pystyä muuttamaan tarvittaessa pelaajakohtaisesti. Vyöhykkeelle pitää myös voida määritellä, onko sen järjestyksellä väliä. Jos järjestyksellä on väliä, tulee vyöhykkeen korttien järjestystä pystyä sekä sekoittamaan että järjestämään. Vyöhykkeiden välillä tulee kyetä siirtämään kortteja.

4.2 Kortin ominaisuuksien määrittely

Ennen toimintojen määrittelyä täytyy vielä alustavasti määritellä yksittäisen kortin perusominaisuudet. Tässä voidaan jälleen hyödyntää Magicia geneeristen osien selvittämiseksi. Magicin Olento-kortti (Creature) koostuu seuraavista tiedoista: kortin yksilöivä nimi (Card name), tyyppi (Type line), tekstikenttä (Text box), resurssikustannus (mana cost), sarjasymboli (Expansion symbol) sekä kyvyt. Nimi, tyyppi ja tekstikenttä ovat listatuista tiedoista geneerisimpiä. Resurssikustannus on myös suhteellisen geneerinen, mutta on olemassa myös suuri joukko korttipelejä, joissa korttien pelaamista rajoitetaan resurssien sijaan muilla tavoin.

Kaikilla korteilla on nimi, jota käytetään kortin yksilöimiseen. Korttipeleissä on yleensä rajoitettu yksittäisten korttien lukumäärää pakassa. Magicissa tämä rajoitus on neljä samannimistä korttia, joskin poikkeuksena sääntöön ovat Basic Land -tyypin kortit. Samannimisten korttien sallittu määrä riippuu yleensä kyseessä olevan korttipelin pakan oletuskoosta. Osa korttipeleistä asettaa pakan koolle vain alarajan, osa sekä ylä- että alarajan.

Osassa on vain yksi sallittu määrä kortteja. Magicin tapauksessa pakan koolle on vain alaraja, joka riippuu pelattavasta formaatista. Pääasiassa pakan alaraja on kuitenkin 60 korttia.

Osa korttipeleistä jakaa kortin nimen kahteen osaan: nimeen ja lisänimeen tai kuvaukseen. Tämä erittely tehdään silloin, kun tarkoituksena on, että kortteja eritellään nimenomaan nimen perusteella. Esimerkiksi Lord of the Rings TCG tulkitsee hahmokortin nimenomaan erisnimen perusteella. Tällöin esimerkiksi *Aragorn, Heir to the Throne of Gondor* ja *Aragorn, King in Exile* lasketaan samaksi kortiksi "*Aragorn*" pelin asettamien korttimäärärajoitteiden kannalta. Geneerisessä toteutuksessa tätä ei tarvitse ottaa erikseen huomioon, sillä kuvausosa voidaan korttipelikohtaisessa toteutuksessa asettaa omaksi muuttujakseen osana korttipelikohtaisen version periyttämistä.

Kortin tyyppi määrittää, miten kortti toimii pelin sääntöjen puitteissa. Magic sisältää korttityypit Land, Artifact, Creature, Instant, Sorcery, Enchantment ja Planeswalker. Käytännössä nämä voidaan jakaa kahteen tyyppiin: efekteihin ja pysyviin (permanent) kortteihin. Nimensä mukaan efektit ovat kertakäyttöisiä kortteja, joilla on jokin vaikutus peliin ja jotka käsittelyn jälkeen laitetaan poistopakkaan. Pysyvät kortit sen sijaan pelataan pelialueelle ja ne pysyvät pelaajan käytettävissä toistaiseksi.

Tekstikenttä jakautuu kahteen osaan: sääntötekstiin ja niin kutsuttuun flavor-tekstiin. Sääntöteksti sisältää nimensä mukaan kortin säännöt eli varsinaiset vaikutukset, mitä kortilla on pelin kulkuun. Sääntöteksti ei välttämättä sisällä kaikkea kortin toimintaan liittyvää: esimerkiksi Magicissa on useita olentokortteja, joiden sääntöteksti on tyhjä: tämän tyyppisillä olennoille ei ole mitään yksilöivää erikoisuutta, vaan ne tarjoavat pelaajan käyttöön vain voimansa ja kestävyytensä. Flavor-tekstillä sen sijaan ei ole pelin kulkuun mitään vaikutusta. Sen tarkoitus on, nimensä mukaan, vain tuoda lisää väriä peliin. Monimutkaisten korttien tapauksessa sääntötekstiltä ei jää tilaa flavor-tekstille, joten myös tämä puoli tekstikenttää voi olla tyhjä.

Resurssikustannus kertoo, mitä resursseja pelaajalla täytyy olla, että hän voi pelata kortin. Magicissa resurssit, mana, jaetaan viiteen magian väriin sekä värittömään manaan. Jotta pelaaja voi pelata kortin, hänen täytyy tuottaa resurssikorteillaan kortissa merkitty määrä värillistä manaa ja/tai merkitty määrä mitä tahansa manaa. Resurssit voivat olla joko pysyviä tai kuluvia. Magic edustaa kuluvia resursseja: pelaaja tuottaa (produce) manaa korttien pelaamiseen pääasiassa Land-tyyppisillä resurssikorteilla, jotka yleensä voivat tuottaa vain yhden resurssin kullakin kierroksella. Pysyvien resurssien tapauksessa resurssit eivät kulu: pelaaja voi pelata kortteja niin kauan kuin hänellä on käytössään vaaditut resurssityypit, resurssien määrästä riippumatta. Yleisin resurssi tämän tyyppisissä peleissä on ryhmittymä: pelaaja voi pelata tiettyyn ryhmittymään liittyviä kortteja vain, jos hänellä on kyseistä ryhmittymää edustava kortti pelialueella.

Kyvyt ovat yleensä numeerinen ilmaisu korttien taistelu- ja muille kyvyille. Magicissa kykyjä on kaksi: voima (power) ja kestävyys (toughness). Voima kertoo, kuinka paljon vahinkoa (damage) olento tekee hyökätessään, ja kestävyys kertoo, kuinka paljon vahinkoa olento kestää ennen kuin se tuhoutuu. Molemmat ovat suhteellisen geneerisiä kykyjä: monissa korttipeleissä Olento-tyyppisillä korteilla ja niiden vastineilla on kestävyys, suurimmassa osassa myös voiman vastine. Aiheutettu vahinko ei kuitenkaan aina toimi samalla tavalla kuin Magicissa. Esimerkiksi Lord of the Rings TCG käyttää järjestelmää, jossa useimmilla korteilla on huomattavasti korkeampi voima kuin kestävyys, mutta oletuksena kortit ottavat hävitessään taistelun vain yhden vahingon. Kaikki korttipelit eivät myöskään käytä kestävyyttä ollenkaan: Legend of Five Rings -korttipelissä taistelun hävinnyt osapuoli menettää kaikki yksittäiseen taisteluun lähettämänsä kortit. Tasapelin tapauksessa molemmat pelaajat menettävät kaikki taistelussa käytetyt kortit.

Sarjasymbolilla ei ole mitään peliteknistä merkitystä. Sitä käytetään ainoastaan ilmaisemaan, mistä sarjasta kyseinen kortti on lähtöisin. Sarjasymbolia tai -tunnusta käytetään vain eri peliformaattien sallittujen korttien tarkastuksissa. Jos kortti on julkaistu useammassa eri sarjassa, täytyy tietää vain uusin sarja, jossa kortti on julkaistu. Formaattien laillisuustarkastukset ovat tärkeitä, sillä Magicia pelataan pääasiassa rajoitetuissa formaateissa. Suosituin Magic-formaatti on standard, jossa laillisia kortteja ovat vain kahden viimeisimmän blokin (yksi iso sarja ja sen pienemmät lisäsarjat) kortit. Standard on suosittu, koska sen metapeli muuttuu tietyin väliajoin uusien blokkien julkaisun myötä, mikä auttaa pitämään pelin tuoreena ja laskee uusien pelaajien aloituskynnystä. Muita tunnettuja formaatteja ovat Vintage, Legacy, Modern ja Extended, jotka kaikki sallivat huomattavasti suuremman korttivalikoiman vaihtelevin rajoituksin pelin tasapainon säilyttämiseksi.

4.3 Toiminnot

Geneerisen korttipelimoottorin suunnittelussa yksi haasteista on tunnistaa kaikki tarvittavat toiminnot erilaisten korttipelien tukemiseksi. Nämä ominaisuudet tulee myös pilkkoa mahdollisimman yksinkertaisiin osiin, sillä pohjimmiltaan suuri osa korttipelien mekaniikoista on samojen toimintojen suorittamista eri vyöhykkeiden välillä. Esimerkiksi kortin nostaminen (draw) ja kortista luopuminen (discard) ovat käytännössä mekaanisesti sama toiminto: kortin siirtyminen yhdeltä pelialueelta toiselle. Kun kortti nostetaan, se siirtyy pakasta pelaajan käsikortteihin. Kun kortista luovutaan, se siirtyy käsikorteista poistopakkaan. Vaadittujen toimintojen löytämiseksi käydään seuraavaksi läpi muutama esimerkkikortti Magicista.

Ensimmäinen esimerkki on kuvassa 3 esiintyvä Lightning Bolt, joka on yksinkertainen punainen kortti. Kortin pelaamisessa on kuitenkin monta vaihetta. Ensimmäiseksi pelaajan täytyy kerätä tarvittavat resurssit, mikä tässä tapauksessa on yksi punainen mana. Tämän jälkeen pelaaja pelaa (play) kortin. Pelaamiseen kuuluu hinnan maksaminen sekä

laillisen (legal) kohteen (target) valinta. Tämän jälkeen kortti laitetaan pinon päällimmäiseksi, josta se käsitellään (resolve) vuorollaan. Kun kortti käsitellään, täytyy tarkistaa paitsi että kohde on edelleen pelissä, myös kohteen laillisuus. Kortin laillisuus tarkistetaan siis kaksi kertaa: pelaamisen ja käsittelyn yhteydessä.

Kuva 3. *Esimerkki yksinkertaisesta kortista. Oikealla ylhäällä näkyy resurssikustannus. Kortin tyyppi on Instant ja efekti 3 vahinkoa yhdelle Olento-kortille tai pelaajalle.*

Lightning Bolt on lisäksi tyypiltään Instant, mikä tarkoittaa, että kortin voi pelata paitsi omalla, myös vastustajan vuorolla, sekä vastauksena muihin toimintoihin. Efekti on kolme vahinkoa. Olennot tuhoutuvat (destroyed), jos ne kärsivät yhden vuoron aikana enemmän vahinkoa kuin mitä niillä on kestävyyttä (toughness). Pelaaja menettää tehdyn vahingon verran elinvoimaa (life points).

Näistä ominaisuuksista saadaan ainakin seuraavat vaatimukset: kohteen valinta, laillisuustarkastukset, resurssien kerääminen ja maksaminen, vahingon aiheuttaminen olennolle ja tuhoutumistarkastelu, pelaajan elinvoiman määrän muuttaminen annetulla arvolla ja häviöehtotarkastelu.

Toinen esimerkki on kuvassa 4 esiintyvä kortti, Mana Leak. Kyseessä on kortti, joka sisältää ehtolauseen. Mana Leak on Instant-tyypin kortti, kuten myös Lightning Bolt. Toisin kuin Lightning Bolt, Mana Leak ei vaikuta pelialueella oleviin kortteihin tai pelaajiin. Sen sijaan Mana Leak vaikuttaa suoraan pinon toimintaan. Magicissa kaikki pelattavat kortit Land-tyypin kortteja lukuunottamatta ovat loitsuja (spell). Estoefekti (counter) tarkoittaa, että kohdekortti poistetaan pinosta ilman käsittelyä. Pinosta poistettu kortti laitetaan poistopakkaan. Mana Leak pelataan kuitenkin samalla tavalla kuin Lightning Bolt ja kaikki muutkin loitsut pelissä: kortin resurssihinta maksetaan ja sille valitaan kohde, minkä jälkeen se asetetaan pinon päällimmäiseksi.

Kuten edellä mainittiin, Mana Leak sisältää ehtolauseen: kohdistetun (targeted) kortin kontrolloija valitsee, maksaako mieluummin kolme väritöntä manaa (jos mahdollista) vai antaako kohdistetun kortin tulla estetyksi. Koska vastustajan käytössä olevat resurssit ovat aina nähtävillä tai helposti pääteltävissä ja koska estoefekti on hyvin voimakas, ei korttia käytännössä juuri pelata, jos vastustaja kykenee valitsemaan mana-vaihtoehdon.

Kuva 4. *Esimerkki kortista, joka vaikuttaa pinoon ja joka sisältää ehtolauseen.*

Näistä ominaisuuksista saadaan ainakin seuraavat vaatimukset: pinosta täytyy voida siirtää kortteja pois ennen kuin niitä käsitellään, pinossa olevia kortteja pitää pystyä valitsemaan kohteeksi, vaikka ne eivät ole vielä pelialueella, ja tarvittaessa täytyy pystyä pyytämään vahvistus pelaajalta, minkä efektin hän valitsee, jos kortilla on vaihtoehtoisia efektejä, kun korttia käsitellään.

Kolmas esimerkki on kuvan 5 kortti. Tällä kertaa on kyseessä kortti, joka asettaa ylimääräisen ehdollisen toiminnon peliin. Abundance on Enchantment-tyypin kortti, mikä tarkoittaa, että se pelataan pelialueelle ja sen vaikutus jatkuu niin kauan kuin kortti pysyy pelialueella. Abundance antaa pelaajalle ehdollisen toiminnon, jonka avulla hän voi korvata normaalin kortin nostamisen toisella toiminnolla. Lisäksi kortti on esimerkki efektistä, joka asettaa kortteja pakan pohjalle pelaajan haluamassa järjestyksessä. Toinen toiminto on korttien paljastaminen pakan päältä ja paljastettujen korttien siirtäminen “tyhjän päälle”: paljastetut kortit ovat väliaikaisesti poissa kaikilta vyöhykkeiltä, eikä niihin voi vaikuttaa millään lailla.

Näistä ominaisuuksista saadaan ainakin seuraavat vaatimukset: korttien paljastaminen pakan päältä, korttien siirtäminen “käsittelyyn”, korttien sijoittaminen pakan pohjalle halutussa järjestyksessä ja ehtolauseen lisääminen olemassa olevaan toimintoon.

Kuva 5. *Esimerkki kortista, joka lisää peliin ylimääräisen ehtolauseen.*

Neljäs esimerkki on hieman monimutkaisempi kortti, jonka käsittelyyn liittyy ehtoja. Kuvassa 6 esiintyvä Dead Ringers eroaa tähän mennessä käsitellyistä korteista siinä, että sillä on monimutkainen ehtolause ja useampi kuin yksi kohde. Dead Ringers kohdistetaan kahteen olentoon, joista kumpikaan ei saa olla väriltään musta. Lisäksi kummallakaan ei saa olla väriä, mitä toisella ei ole. Kyseessä on siis korttien värien exact-match -tarkistus. Myöskään viimeistä sääntölauseketta ei sovi unohtaa: kyseessä on efekti, joka estää toisen olemassa olevan efektin, tässä tapauksessa uusiutumisen (regenerate) toiminnan.

Kuva 6. *Esimerkki kortista, jolla on hiukan monimutkaisempi ehtotarkistelu.*

Värillä Magicissa viitataan pelin viiteen magian väriin: punaiseen, vihreään, siniseen, mustaan ja valkoiseen. Suurin osa pelin korteista on yksivärisiä, mutta lisäksi on olemassa

monivärisiä (multi-color) kortteja. Näiden tapauksessa kortilla on kaikki ne värit, joita kortin maksamiseen tarvitaan. Esimerkiksi kortti, jonka resurssikustannus on yksi punainen, yksi musta ja kolme väritöntä manaa olisi väriltään sekä musta että punainen. Samalla tavalla kortti, jonka resurssikustannus on yksi punainen tai musta ja kaksi väritöntä olisi sekä punainen että musta. Lisäksi pelissä on kortteja, jotka muuttavat korttien väriä joko pysyvästi tai väliaikaisesti.

Dead Ringers on myös esimerkki kortista, jonka kohdalla on merkityksellistä, että ehto tarkistetaan uudelleen korttia käsiteltäessä. Jos kumpikaan kortti ei enää ole laillinen kohde, kortin efekti estyy ja kortti siirtyy poistopakkaan. Jos vain toinen kohde on laillinen, tarkistetaan molemmilta kohteilta värien yhteneväisyys, mutta tuhoutumisefekti vaikuttaa vain lailliseen kohteeseen. Jos toinen kohteista poistuu pelialueelta ennen efektin tarkistusta, käytetään tarkistuksessa väriä, mikä kohteella oli ennen sen poistumista.

Näistä ominaisuuksista saadaan ainakin seuraavat vaatimukset: tyyppitarkastelut osana laillisuustarkastelua, useamman kohteen tarkistukset, pelikohtaisesti määriteltävät laillisuustarkastuspisteet, exact-match -tarkastelut ja toisen, olemassa olevan efektin estäminen.

Viides tarkasteltava esimerkki on kuvan 7 kortti, Opalescence, joka vaikuttaa muiden korttien tyyppiin. Opalescence on Enchantment-tyypin kortti, kuten aiemmin käsitelty Abundance. Se tekee kaikista muista Enchantment-tyypin korteista lisäksi Olento-tyypin kortteja. Kortit saavat Olento-korttien tapaan statistiikat, mitä niillä ei normaalisti ole. Lisäksi Enchantment-kortit saavat kaikki Olento-korttien lisäsäännöt ja rajoitukset.

Kuva 7. *Esimerkki kortista, joka vaikuttaa muiden korttien tyyppiin.*

Tästä ominaisuudesta voidaan johtaa ainakin se, että korteille pitää pystyä lisäämään ja poistamaan dynaamisesti tyyppejä. Lisäksi esimerkiksi Enchantment – Aura -tyypin kortit eivät olennoiksi muututtuaan pysty toimimaan normaalilla tavalla, koska Olento-tyypin kortit eivät voi kiinnittyä toisiin kortteihin. Lisäksi Aura-tyypin kortti automaattisesti tuhoutuu ja siirtyy poistopakkaan, koska se ei enää kykene täyttämään attachment-vaatimustaan. Dynaamisten tyyppimuunnosten kohdalla täytyy siis suorittaa erikseen laillisuustarkasteluja.

Viimeinen käsiteltävä esimerkki on kortti, joka luo muista korteista kopioita, eli generoi efektejä, joille ei ole omaa korttia. Kuvassa 8 esiintyvä Eye of the Storm on esimerkki hieman monimutkaisemmasta kortista. Käytännössä se kasaa itseensä Instant- ja Sorcery-tyypin kortteja sitä mukaa, kun peli etenee, mikä pidemmän päälle johtaa hyvin monimutkaisiin ketjureaktioihin. Huomion arvoista kortin sanoituksessa on termi card: Eye of the Storm ei aktivoidu omista kopioistaan, sillä kopiot eivät ole varsinaisia kortteja.

Kuva 8. *Esimerkki kortista, joka luo uusia efektejä.*

Korttien kopiot menevät pinoon aivan kuten kortit normaalisti. Pinossa ollessaan kopio myös toimii kuin normaali kortti: se voidaan estää, joskin jokainen kopio vaatii oman estoefektinsä, ja siihen voidaan vaikuttaa normaalisti. Kun kopio on käsitelty, se poistuu pinosta ja lakkaa olemasta.

Eye of the Storm on myös esimerkki kortista, joka sisältää triggerin, eli ehdon, josta seuraa jokin efekti. Eye of the Storm toimii lisäksi kaikille pelaajille: se ei teoriassa millään lailla suosi pelaajaa, joka sen alun perin pelasi. Käytännössä tämä tietenkin otetaan huomioon pakan suunnitteluvaiheessa, ja pelaaja pyrkii maksimoimaan oman ja minimoimaan vastustajan saaman hyödyn.

Näistä ominaisuuksista saadaan ainakin seuraavat vaatimukset: olemassa olevien ehtojen tarkistaminen jokaisen tapahtuman jälkeen, korttien efektien kopioiminen pinoon ja viittaukset kortista yhteen tai useampaan muuhun korttiin.

4.4 Vuororakenne

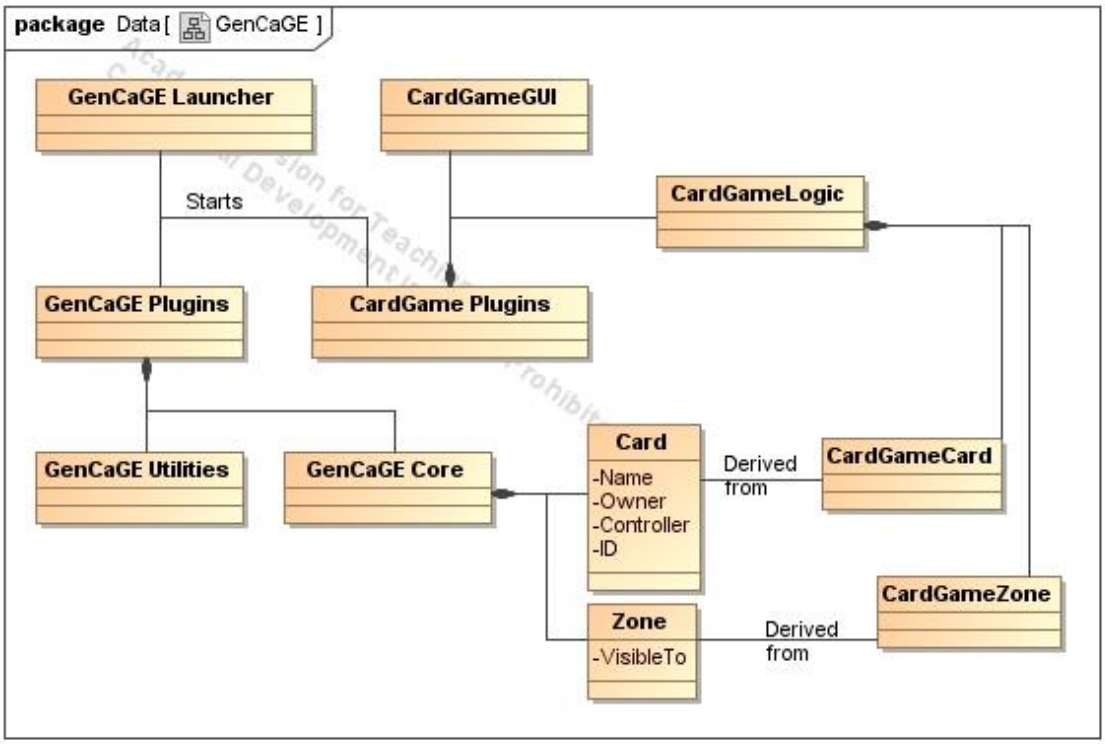
Jokaisella korttipelillä on omanlaisensa vuororakenteen. Tämän vuoksi myös jokaisen korttipelikohtaisen toteutuksen täytyy toteuttaa oma vuororakenteensa. Vuororakenne koostuu listasta vaiheita ja alavaiheita. Magicia esimerkkinä käyttäen vaiheet olisivat *Beginning*, *Main 1*, *Combat*, *Main 2* ja *End*. Päävaiheiden lisäksi tarvitaan tuki useamman alavaiheen luomiseksi. Jokaiseen vaiheeseen ja sen osaan täytyy lisäksi voida määrittää, kuka voi ja minkä tyyppisiä kortteja ja efektejä siinä on mahdollista pelata. Esimerkiksi Magicissa vain vuorossa oleva pelaaja voi pelata *Sorcery*-tyypin kortteja *Main 1*- ja *Main 2*-vaiheissa, mutta kumpikin pelaaja voi pelata *Instant*-tyypin kortteja. Joissain vaiheissa tai osissa ei välttämättä ole mahdollista pelata mitään kortteja tai efektejä. Jokaiseen vaiheeseen tai sen osaan voi lisäksi liittyä erilaisia toimintoja. Yleisin geneerinen esimerkki tästä on yhden tai useamman kortin nostaminen, eli korttien siirtäminen pakasta käsikortteihin.

Itse vaiheisiin täytyy voida sitoa korttien asettamia ehtolauseita. Ehtolauseita on kahdenlaisia: tietyssä vaiheessa tai sen osassa laukeava efekti ja efekti, joka jatkuu aktivoitumisestaan tiettyyn vaiheeseen asti, yleensä kyseisen vuoron loppuun asti. Esimerkkinä ensimmäisestä mainittakoon Magicissa yleinen tapaus: efekti, joka laukeaa *Beginning*-vaiheen *Upkeep*-osan alussa niin kauan kuin efektin laukaiseva kortti on pelialueella. Jälkimmäisen ehtolausetyypin kohdalla täytyy ottaa huomioon, että tiettyyn vaiheeseen tai sen osaan asti jatkuvat efektit täytyy siivota pois ja mahdolliset vaikutukset pelialueella oleviin kortteihin tarkistaa. Useamman yhtäaikaisen efektin tapauksessa täytyy ottaa huomioon pelikohtainen efektien järjestyksen määrittämistapa. Yleensä vuorossa oleva pelaaja valitsee suoritusjärjestyksen. Vaiheeseen liittyvät efektit käsitellään erikseen, ja niillä on oma, ennalta määrätty suoritusjärjestyksensä. Esimerkkinä tästä on Magicin *End*-vaiheen *Cleanup*-osa, jossa kaikilta Olento-korteilta poistetaan vuoron aikana mahdollisesti kertynyt vahinko ja *Until End of Turn* -efektit siivotaan pois. Järjestyksellä on tässä väliä, koska efektien siivouksen jäljiltä Olento-kortilla saattaisi olla enemmän vahinkoa kuin kestävyyttä, jolloin sen tulisi normaalisti tuhoutua.

5. ARKKITEHTUURISUUNNITTELU JA TOIMINNOT

Tässä luvussa käydään läpi geneerisen korttipelimoottorin arkkitehtuurisuunnittelua ja toimintoja. Luvussa keskitytään korttipelimoottorin tietorakenteisiin, mutta selkeyden vuoksi esitellään aluksi myös koko sovelluksen karkea rakenne.

Kuvassa 9 esitellään sovelluksen käsitekaavio. Mayerin mallin mukaisesti sovelluksen pääsovellus on minimaalinen, sillä pääosa sovelluksesta on toteutettu liitännäisinä. Korttipelin käynnistyksen yhteydessä sovellus käynnistää vain kyseisen korttipelin tarvitsemat liitännäiset. Kaaviossa esitellyt liitännäiset ovat suuntaa antavia. GenCaGE Utilities -liitännäinen kuvassa edustaa kaikkia pienempiä liitännäisiä, joita lopullinen sovellus saattaa tarvita. Vastaavasti GenCaGE Core -liitännäisen alle kuuluu sekä varsinainen pelitila että muut lopullisen sovelluksen ydinosat kuten pakanrakennusmoodi. Korttipelikohtaiset liitännäiset, eli kuvan CardGameGUI ja CardGameLogic, noudattavat työssä kehitetyn esimerkkikorttipelin jakoa. Sovellus ei ota kantaa korttipelikohtaisen toteutuksen liitännäisten kokoon tai määrään. Pääsovellus käynnistää yksittäisen korttipelin sen asetustiedostossa määritellyillä liitännäisillä.



Kuva 9. *Sovelluksen käsitekaavio.*

5.1 Arkkitehtuuriratkaisut

Työn kannalta oleellisimmat osat korttipelimoottoria ovat varsinainen pelitila ja sen päälle rakennettava sääntökerros, joka valvoo pelin kulkua. Pelitilan sisäinen toiminnallisuus kuuluu geneerisen toteutuksen puolelle, mutta sääntökerros jakautuu toteutuksen kannalta kahteen osaan: korttipelikohtainen toteutus vastaa pelin sääntöjen tulkinnasta, geneerisen osuuden vastuulle jää tarjota riittävän kattava toimintojen määrä, että se mahdollistaa pelin sääntöjen toteuttamisen. Geneerisen tietorakenteen täytyy esimerkiksi tukea pelaajien resurssien seuraamista ja korttien siirtymistä vyöhykkeeltä toiselle.

Työssä ei oteta kantaa kaikkiin lopullisen sovelluksen kannalta oleellisiin osiin. Esimerkkeinä työn kattavuuden ulkopuolelle karsituista ominaisuuksista mainittakoon pakanrakennustila ja moninpeli verkon ylitse. Pakanrakennustilan pääasiallinen tarkoitus on luoda yksinkertainen ja helppokäyttöinen käyttöliittymä, jonka avulla voidaan luoda ja täyttää tietorakenne, jonka pelitila tulkitsee pakaksi. Verkon ylitse pelaaminen puolestaan helpottaa huomattavasti sovelluksen päällä pyörivien korttipelien testaamista ja esittelyä.

Työn kannalta oleellisessa osassa on arkkitehtuurin geneerisyys. Koska korttipelimoottorin on tarkoitus olla ottamatta kantaa korttipeleihin, joita sillä pyöritetään, on sen kyettävä venymään hyvin erilaisten korttipelien vaatimuksiin.

Geneerisyys toteutuu työssä mukailemalla Mayerin liitännäispohjaista sovelluskehitysmallia. Lisäksi työssä hyödynnetään periyttämistä. Liitännäisten käytettäväksi toteutetaan kirjasto yksinkertaisia toimintoja, joita yhdistelemällä on mahdollista toteuttaa myös monimutkaisia tapahtumia. Seuraavaksi käydään lävitse sovelluksen eri osa-alueita ja miten edellä mainittuja ratkaisuja on tarkoitus hyödyntää niiden tapauksessa. Osa-alueisiin kuuluvat pelaajat ja resurssit, vyöhykkeet, kortit ja vuororakenne.

5.2 Pelaajat ja resurssit

Pelaajan ominaisuudet toteutetaan korttipelimoottorissa hyödyntämällä luokkaa *Resource*, jonka kautta voidaan toteuttaa kaikki pelaajan resurssit ja ominaisuudet. Magicia esimerkkinä käyttäen resursseja ovat esimerkiksi elinvoima ja jokainen kuudesta manatyypistä – viisi magian väriä ja väritön mana. Resursseilla voidaan myös tarvittaessa esittää numeerisena ei-numeerisia käsitteitä kuten Legend of Five Rings -korttipelin klaanit, jolloin tietty numeerinen arvo viittaa tiettyyn klaaniin. Resurssilla on nimi ja numeerinen arvo. Resursseja varten toteutetaan yksinkertaiset numeerista arvoa muokkaavat toiminnot. Resurssit toteutetaan pelaajaan liittyvänä HashMap-tyyppinä, jossa resurssin nimi toimii avaimena ja itse resurssi esitetään numeerisena arvona. Geneerinen toteutus ei ota kantaa näihin resursseihin, mutta pelaaja tarjoaa funktiot arvojen lisäämiseen, muuttamiseen ja hakemiseen.

Resurssien lisäksi pelaajaan liittyy myös vyöhykkeitä. Osa neutraaleista vyöhykkeistä kuten Magicin *Battlefield* ja *Exile* voidaan jakaa kahtia ja liittää selvyyden vuoksi yksittäiseen pelaajaan. Tämä helpottaa etenkin korttipelin graafista esittämistä, kun kortit liittyvät selkeästi yksittäiseen pelaajaan. Toisaalta geneerisyyden vuoksi vyöhyke on myös mahdollista liittää useampaan pelaajaan. Tämän myötä korttipelimoottori tukee myös korttipelejä, joissa pelaajilla on esimerkiksi yhteinen pakka ja poistopakka. Vyöhykkeet toteutetaan HashMap-tyyppinä, jossa vyöhykkeen nimi toimii avaimena.

Korttipelissä on yleensä kaksi tai enemmän pelaajia. Tämän työn puitteissa oletetaan, että pelaajia on kaksi, mutta suunnittelussa otetaan huomioon myös useamman pelaajan mahdollisuus.

5.3 Vyöhykkeet

Perusvyöhykkeet korttipelimootorissa ovat pakka, poistopakka ja käsikortit. Näiden oletetaan olevan käytössä kaikissa korttipeleissä ja niille toteutetaan niihin liittyvät oletustoiminnot. Muut vyöhykkeet, kuten Magicin poistoalue, toteutetaan periyttämällä korttipelikohtaisen liitännäisen puolella. Kaikki vyöhykkeet joko luodaan geneerisen *Zone*-luokan ilmentyminä tai periytetään siitä. *Zone*-luokka määrittelee muutaman perustoiminnon, jotka kaikkien vyöhykkeiden tulee toteuttaa. Näihin kuuluu muun muassa kortin antaminen ja vastaanottaminen vyöhykkeelle.

Vyöhykkeet joko liittyvät yksittäiseen pelaajaan tai ovat neutraaleja. Osa vyöhykkeistä voi myös olla korttipelin sääntöjen näkökulmasta neutraaleja, mutta toteutuksen puolella liittyä pelaajaan teknisistä syistä. Esimerkkinä tällaisesta voidaan käyttää Magicin pelialuetta, joka on järkevä jakaa kahteen osaan pelaajien kesken pelialueen graafisen esittämisen toteutuksen helpottamiseksi.

5.4 Kortit

Kortit ovat eniten variaatiota sisältävä osa korttipelimoottoria. Siksi kortit toteutetaan periyttämällä abstraktista kantaluokasta Card. Card sisältää geneerisessä muodossaan suhteellisen vähän informaatiota, sillä korttien tietosisältö voi vaihdella merkittävästi korttipelien välillä. Abstraktin Card-luokan pääasiallinen funktio onkin toimia geneerisenä pohjana, jota sovellus voi käsitellä ottamatta kantaa siihen, miten yksittäinen korttipeli toimii. Tämän vuoksi yksittäisestä kortista ei kannata geneeriselle puolelle tallettaa kuin seuraavat asiat: kortin ID, nimi, timestamp, omistaja sekä kontrolloija. Loppu informaatio lisätään korttiin periyttämällä. Kentistä ID ja nimi auttavat yksilöimään kortin. Timestamp-kentän kautta selvitetään esimerkiksi se, missä järjestyksessä kortit ovat siirtyneet pelialueelle. Tämä tieto on tärkeä harvinaisissa tilanteissa, joissa ajoituksella on merkitystä useamman vuoron aikana. Omistaja on arvoltaan pysyvä kenttä, joka viittaa pelaajaan, jonka pakasta kortti on lähtöisin. Kontrolloija on yleensä sama kuin omistaja, mutta

useissa korttipeleissä on efektejä, jotka siirtävät kortin kontrollin pelaajalta toiselle. Magicin tapauksessa kortin omistaja on oleellinen esimerkiksi silloin, kun kortti siirtyy pelialueelta poistopakkaan. Tällöin kortti siirtyy kontrolloijasta riippumatta omistajansa poistopakkaan.

Jokainen korttipeli toteuttaa omat korttinsa periyttämällä käyttämänsä eri korttityypit Card-luokasta. Geneerinen toteutus käsittelee kaikkia kortteja Card-luokan edustajina. Korttien toiminnot ja niiden käsittely suoritetaan korttipelikohtaisen toteutuksen puolella, hyödyntäen korttipelimoottorin tarjoamia geneerisiä funktioita.

5.5 Vuororakenne

Korttipeleillä on erilaisia vuororakenteita. Pääasiassa ne voidaan jakaa kahteen päätyyppiin: vuorollisiin ja kierroksellisiin. Vuorollisissa korttipeleissä vuororakenne koostuu useista vaiheista. Jokainen pelaaja käy vuorollaan koko vuororakenteen lävitse, minkä jälkeen vuoro siirtyy seuraavalle pelaajalle. Esimerkiksi Magic on vuorollinen korttipeli.

Myös kierroksellisissa vuoropeleissä vuororakenne koostuu vaiheista, mutta sen sijaan, että yksi pelaaja kerrallaan käy koko vuororakenteen lävitse, kaikki pelaajat suorittavat yhden vaiheen toiminnot vuorotellen ennen seuraavaan vaiheeseen siirtymistä. Kun kaikki pelaajat ovat käyneet kaikki vaiheet lävitse, alkaa uusi kierros.

Molemmat päätyypit voidaan ottaa huomioon jakamalla korttipelin vuororakenne hierarkisesti osiin: korkeimpana käsitteenä on “kierros” (*Round)*. Kierroksen alapuolella on “vuoro” (*Turn*). Vuoron alle määritellään “vaiheet” (*Phase*) ja vaihekohtaiset “askeleet” (*Step*). Joissain tapauksissa “askelilla” voi olla vielä “ala-askelia”. Oletuksena käytetään Magicin jakoa näihin neljään käsitteeseen.

Vuorollisten korttipelien vuororakenne toteutetaan yksinkertaisesti luomalla kierroksen alle yksi vuoro per pelaaja. Nämä vuorot sisältävät koko vuororakenteen kaikki vaiheet askeleineen. Kierrokselliset korttipelit vastaavasti sisältävät jokaista vaihetta varten vuoron jokaista pelaajaa kohden. Jokainen vuoro sisältää tietyn vaiheen ja vaiheen askeleet. Vuororakenne koostuu siis geneerisistä osista, joista korttipelikohtainen toteutus rakentaa itselleen vaatimansa kokonaisuuden.

5.6 Toimintakirjasto

Toimintakirjaston pääasiallinen tehtävä on toimia yhteytenä korttipelimoottorin tietokannan ja korttipelikohtaisen toteutuksen välillä. Sen toiminta voidaan jakaa kahteen osaan: tiedon välittämiseen tietokannalta korttipelitoteutuksen graafiselle käyttöliittymälle ja geneeristen toimintojen tarjoamiseen korttipelikohtaiselle logiikalle.

6. ESIMERKKIKORTTIPELIN MÄÄRITTELY

Tässä luvussa määritellään yksinkertainen korttipeli, jonka toteutuksen prototyyppi toteutetaan esimerkkinä geneerisen korttipelimoottorin toiminnasta. Korttipelin on tarkoitus olla riittävän monipuolinen sovelluksen kattavuuden esittelyyn, mutta tarpeeksi yksinkertainen, että sen toteutus ei turhaan kasvata työn laajuutta.

6.1 Korttipelin kuvaus ja tavoite

Noudattaen korttipelien kaksintaistelua painottavaa perinnettä on esimerkkikorttipeli yksinkertainen simulaatio kaksintaistelusta geneeriseen fantasiamaailmaan sijoittuvassa areenassa. Teeman mukaisesti korttipelin nimeksi annetaan ”Areena”. Molemmilla pelaajilla on gladiaattori-kortti joka edustaa pelaajaa itseään. Pelin alussa molemmat gladiaattorit ovat aseistamattomia. Pelin kulun myötä gladiaattorit voivat napata maasta varusteita, hyökätä vastustajansa kimppuun, suojautua vastustajan hyökkäyksiltä ja hyödyntää erilaisia taktiikoita ja taistelutaitoja. Kaikkea tätä kuvataan tietenkin korteilla.

Pelin tavoitteena on olla viimeinen pystyssä oleva gladiaattori. Pelaaja häviää pelin, jos hänen gladiaattorinsa menettää tajuntansa kärsittyään liian paljon vahinkoa tai jos gladiaattorin kunto pettää eikä tämä jaksa enää taistella, mikä tapahtuu jos pelaajan yrittää nostaa kortin tyhjästä pakasta.

6.2 Korttityypit

Korttipeleille tyypilliseen tapaan myös Areena-korttipeli sisältää sekä pelin pelialueille pelattavia kortteja että kortteja, joilla on kertaluokkainen vaikutus pelitilanteeseen. Edelliseen kategoriaan kuuluvat Gladiaattori-, Varuste- ja Taktiikka -tyypin kortit, jälkimmäiseen Toiminto-tyypin kortit.

Gladiaattori: Gladiaattori, jonka roolin pelaaja ottaa pelissä. Jokaisella gladiaattorilla on kestävyys, hyökkäys ja taito. Kestävyys määrää, kuinka paljon vahinkoa gladiaattori kestää ennen tajunnan menetystä. Hyökkäys on yksinkertainen liike, jonka gladiaattori voi käyttää, jos ei pysty tai halua käyttää Hyökkäys-korttia. Taito on jokaiselle gladiaattorille ominainen erikoistaito, joka erottaa hänet muista gladiaattoreista.

Varuste: Varusteet ovat erilaisia aseita ja muita apuvälineitä, joita gladiaattorit voivat löytää taistelukentältä. Varusteilla voi olla joko pysyvä vaikutus, esimerkiksi bonus kaikkiin hyökkäyksiin, tai kertakäyttöinen hyöty, minkä jälkeen varuste siirtyy poistopakkaan. Osa varusteista tarvitsee yhden tai molemmat gladiaattorin käsistä että niitä voidaan käyttää. Jos gladiaattorilla ei ole riittävästi vapaita käsiä, voi pelaaja siirtää käytössä olevia varusteitaan poistopakkaan käsien vapauttamiseksi.

Taktiikka: Taktiikka-kortit kuvaavat erilaisia taistelutaktiikoita, joita gladiaattori voi hyödyntää taistelussa. Jokainen Taktiikka-kortti antaa pysyvän edun käyttäjälleen niin kauan kuin se pysyy pelissä. Jokaisella taktiikalla on jokin ehto, jolla vastustaja voi poistaa kortin pelistä. Pelaajalla voi olla kerralla vain yksi aktiivinen Taktiikka-kortti. Pelaaja voi poistaa oman Taktiikka-kortin pelistä pelaamalla uuden Taktiikka-kortin.

Toiminto: Toiminnot ovat hyökkäyksiä, puolustuksia ja reaktiota, joita gladiaattorit voivat tehdä taistelussa. Toiminnot pelataan pinoon, mistä ne käsitellään. Kun toiminto on käsitelty, se siirtyy poistopakkaan. Toiminto-kortilla voi olla jokin esiehto, esimerkiksi tietty aktiivinen Varuste-kortti, jonka täytyy toteutua, että kortin voi pelata.

6.3 Vuororakenne

Esimerkkikorttipelin on tarkoitus olla yksinkertainen, minkä tulee myös näkyä sen vuororakenteessa. Kuvassa 10 esitellään pelin vuororakenne.

Kuva 10. *Esimerkkikorttipelin vuororakenne. Vaiheet kulkevat vasemmalta oikealle, vaiheen osat eli askeleet ylhäältä alas.*

Refresh-vaiheessa vuorossa oleva pelaaja voi ensiksi siirtää yhden käsikorttinsa poistopakkaan (Discard) ja tämän jälkeen täyttää käsikorttinsa viiteen korttiin (Draw). Jos pelaaja ei pysty täyttämään käsikorttejaan viiteen, koska hänen pakkansa on loppu, häviää hän pelin.

Plan-vaiheessa pelaaja voi joko pelata yhden Varuste- tai Taktiikka-kortin kädestään tai poistaa vastustajan Taktiikka-kortin täyttämällä sen ehdon.

Assault-vaiheessa vuorossa oleva pelaaja voi joko pelata yhden hyökkäystyypin Toiminto-kortin tai käyttää gladiaattorinsa oletushyökkäystä (Attack). Oletushyökkäyksen käyttäminen vaatii yhden käsikortin siirtämistä poistopakkaan. Hyökkäyksen valitsemisen jälkeen kumpikin pelaaja voi pelata vuorotellen muita toimintoja pinoon vastauksena

hyökkäykseen ja muihin toimintoihin (Reactions). Toiminnot käsitellään pinosta yksi kerrallaan, ja jokaisen käsitellyn kortin jälkeen molemmilla on tilaisuus pelata uusi reaktio. Kun kaikki reaktiot on käsitelty, pelaajat laskevat, paljonko vahinkoa hyökkäys tekee. Tehty vahinko vähennetään puolustavan gladiaattorin kestävyydestä (health).

Regroup-vaiheessa kumpikin laskee käsikorttiensa määrän viiteen (Cleanup) siirtäen mahdolliset ylimääräiset kortit poistopakkaan. Lopuksi kaikki vuoron loppuun asti vaikuttavat efektit loppuvat ja vuoro päättyy (End).

6.4 Pelin kulku

Esimerkkikorttipelin ymmärtämisen helpottamiseksi annetaan tässä esimerkki pelin kulusta. Pelaajina ovat Matti ja Mikko. Alkutilanteessa Matin vuoro on alkamassa. Matilla on kaksi käsikorttia ja juuri vuoronsa lopettaneella Mikolla neljä.

Refresh-vaiheessa ensimmäinen askel on Discard. Matin täytyy ensimmäiseksi valita, haluaako hän karsia toisen kahdesta käsikortistaan. Matti toteaa, että hänen käsikorteissaan ei ole yhtään hyökkäykseen sopivaa Toiminto-korttia. Tämän perusteella Matti päättää karsia käsikortin kasvattaen ylimääräisen noston myötä todennäköisyyttä, että pakasta löytyisi sopiva Toiminto. Tämän jälkeen Matti siirtyy Draw-askeleeseen ja nostaa neljä korttia kasvattaen käsikorttien määrän jälleen viiteen. Matin iloksi pakasta nousee neljä Toiminto-korttia, joista kaksi aloittaa hyökkäyksen.

Plan-vaiheessa Matti huomioi, että Mikolla on pelissä Taktiikka-kortti, joka merkittävästi vähentää Matin hyökkäyksien aiheuttamaa vahinkoa. Taktiikan poistamisen ehtona on, että vuorossa oleva pelaaja siirtää kaksi kädessä olevaa Toiminto-korttia poistopakkaan. Koska Matilla on kädessä useita Toiminto-kortteja, hän päättää poistaa Mikon Taktiikka-kortin pelistä täyttämällä annetun ehdon. Matilla on jäljellä kolme käsikorttia.

Assault-vaiheessa Matti valitsee käsikorteistaan hyökkäystyypin Toiminto-kortin. Puolustavana pelaajana Mikko valitsee ensin reaktionsa. Ensimmäisenä reaktionaan Mikko pelaa toiminnon, joka nostaa hänelle kolme ylimääräistä korttia, mutta kasvattaa Matin tekemän vahingon määrää. Molemmat pelaajat jättävät reagoimatta ja Mikon kortti käsitellään. Käsittelyn jälkeen pelaajat saavat uuden mahdollisuuden reagoida. Uusien käsikorttien myötä Mikko jatkaa puolustustaan pelaamalla puolustustyypin Toiminto-kortin, joka laskee hyökkäyksen aiheuttaman vahingon nollaan. Matin onneksi toinen hänen jäljellä olevista korteistaan on reaktiotyyppinen toiminto, joka mitätöi yhden vastustajan puolustustyypin toiminnon. Mikko vastaa pelaamalla toisen kappaleen aiemmasta kortteja nostavasta toiminnosta. Matilla ei ole tilanteeseen sopivaa Toiminto-korttia kädessä eikä hän siksi voi reagoida. Mikon kolmesta lisäkortista huolimatta hänellä ei ole tilanteeseen sopivaa Toiminto-korttia. Täten Mikon puolustuksen mitätöivä toiminto jää voimaan. Tämän lisäksi Mikko on onnistunut kahdesti nostamaan hyökkäyksen aiheuttamaa

vahinkoa. Pelaajat laskevat Matin hyökkäyksen lopullisen vahingon ja Mikko vähentää tuloksen gladiaattorinsa kestävyydestä.

Lopuksi Matti siirtyy Regroup-vaiheeseen. Cleanup-askeleessa pelaajat tarkistavat käsikorttiensa määrän ja Mikko toteaa, että hänellä on kädessään seitsemän korttia. Mikko siirtää kädestään kaksi korttia poistopakkaan. Viimeisenä osana Matin vuoroa on End-askel. Koska kummallakaan pelaajalla ei ole väliaikaisia efektejä pelissä, tässä askeleessa ei tapahdu mitään ja Matin vuoro loppuu. Matin hyökkäys onnistui yli odotusten, mutta nyt hänen tulisi selvitä seuraavaan vuoroonsa asti vain yhdellä käsikortilla.

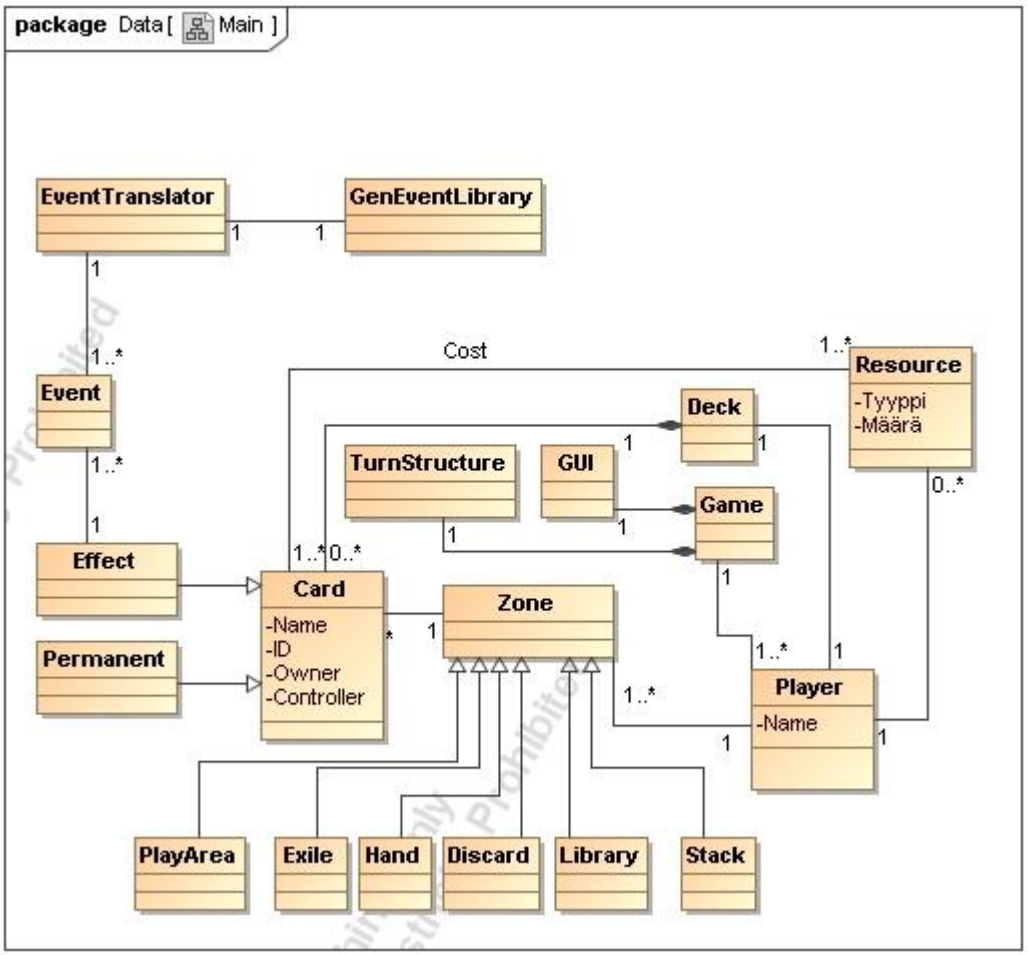
Peli etenee tästä tilanteesta eteenpäin vuoron kiertäessä pelaajalta toiselle. Peli päättyy kun toisen pelaajan gladiaattorilta loppuu kestävyys tai pelaaja ei enää voi nostaa kortteja pakan loppumisen vuoksi.

7. KORTTIPELIMOOTTORIN TOTEUTUS

Tässä luvussa käsitellään työn osana toteutetun korttipelimoottoriprototyypin edistymistä sekä vastaan tulleita ongelmia ja niiden ratkaisuja. Osana korttipelimoottorin kehitystä oli toteutus myös edellisessä luvussa esitellylle esimerkkikorttipelille.

7.1 Arkkitehtuurisuunnittelu

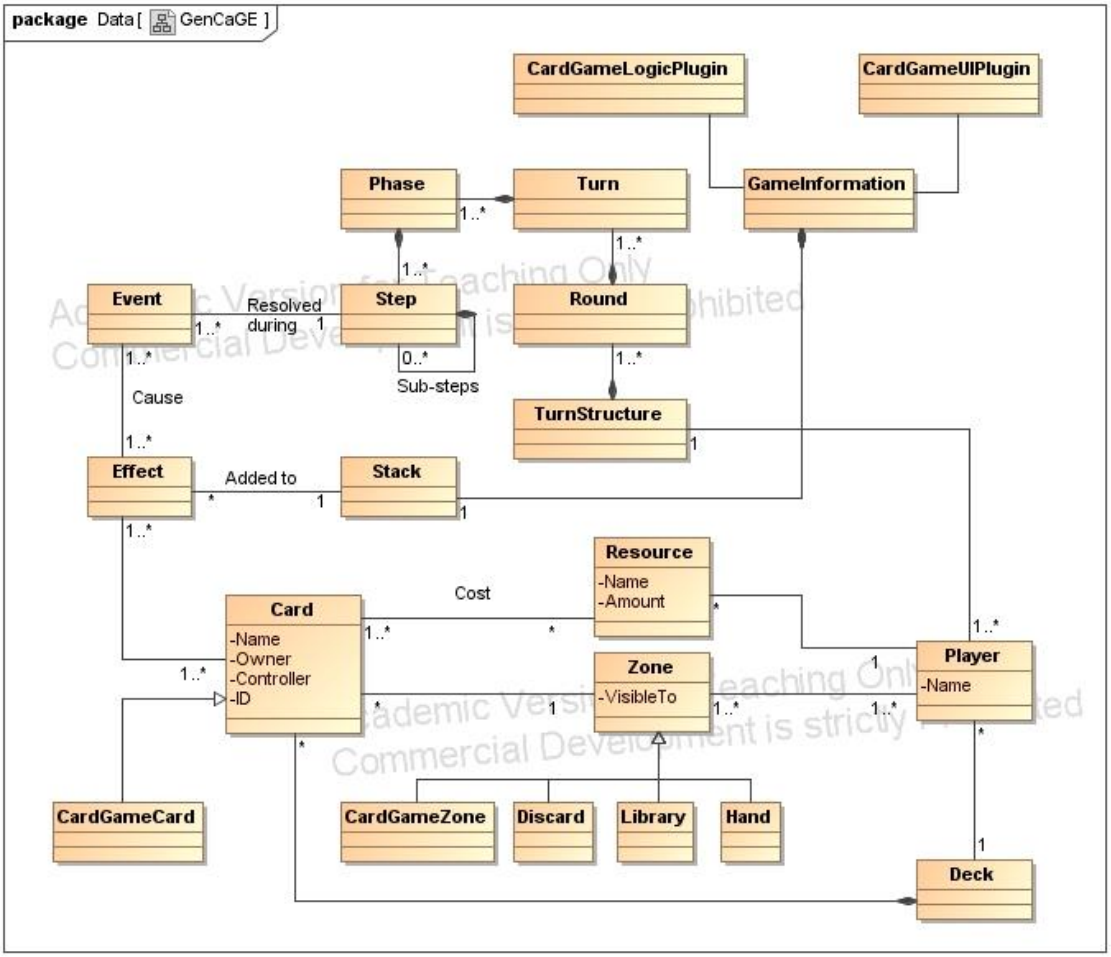
Geneerisen korttipelimoottorin määrittely on käsitelty aiemmissa luvuissa. Myös lopullinen arkkitehtuuri on käsitelty aiemmin. Tässä kohdassa sen sijaan tarkastellaan arkkitehtuurin kehitystä projektin edistyessä painottaen geneerisen korttipelimoottorin kannalta oleellista tietorakennetta.



Kuva 11. *Ensimmäinen konkreettinen luokkakaavio arkkitehtuurista, 18.10.2013.*

Kuvassa 11 on ensimmäinen työstetty versio sovelluksen luokkakaaviosta. Keskeisiä käsitteitä ovat Zone, Card ja Player. Sovelluksen Game-käsite on hyvin pienessä osassa koostuen vain graafisesta käyttöliittymästä (GUI) ja vuororakenteesta (TurnStructure).

Työn edetessä luokkakaaviosta karsittiin ylimääräisiä käsitteitä pois ja tarkennettiin toisia.



Kuva 12. *Viimeisin luokkakaavio kuva arkkitehtuurista, 4.9.2014.*

Kuvassa 12 on viimeinen työtä varten kehitetty arkkitehtuurikuva. GUI-käsite on siirretty liitännäisen vastuulle ja myös korttipelikohtainen logiikka esiintyy omana liitännäisenään. Ensimmäinen iso huomio on, että joitain alkuperäisen luokkakaavion käsitteitä siirrettiin korttipelikohtaisten liitännäisten puolelle korttipelien monimuotoisuuden vuoksi. On mahdotonta ennustaa, mitä kaikkea korttipeli käyttöliittymältään vaatii, joten on parempi jättää korttipelikohtainen käyttöliittymä osaksi korttipelikohtaista toteutusta. Vastaavasti ei ole järkevää yrittää ottaa kaikkia olemassa olevia kortteja huomioon geneerisen toteutuksen puolella. Vaikka suuri osa korteista voidaan toteuttaa yhdellä yksittäisellä funktiolla geneerisellä puolella, on olemassa hyvin monimutkaisia kortteja, jotka on järkevämpi jättää pääosin korttipelikohtaisen logiikan käsiteltäväksi. Tällöin korttipelikohtainen logiikka suorittaa itse pääosan kortin toiminnoista ja välittää geneeriselle puolelle vain toimintojen lopputulokset.

Toinen merkittävä ero on, että *Card*-luokkaa ei ole jaettu Permanent- ja Effect-alaluokkiin. Jaottelun tarkoitus oli ollut, että sen kautta olisi voinut ohittaa tiettyjä tarkasteluja. Geneerisen toteutuksen periaatteen vuoksi päädyttiin kuitenkin jättämään kaikki korttien periytys korttipelikohtaisen toteutuksen puolelle.

Kolmantena huomiona on *TurnStructure*-luokan tarkempi jaottelu ja *Step*-luokan yhteys *Event*- ja *Effect*-luokkien kautta kortteihin. Käytännössä tämä tarkoittaa, että kortit luovat efektejä, joista yksi mahdollisuus on myöhemmin tapahtuvien tapahtumien luonti. Kortissa voi esimerkiksi olla kaksi efektiä: heti tapahtuva efekti “3 vahinkoa vastustajalle” sekä ajastettu efekti “nosta 1 kortti seuraavan vuoron *Draw*-vaiheen aluksi”.

7.2 Toteutuksen ongelmat

Tässä kohdassa käsitellään toteutuksessa vastaan tulleita ongelmia ja niihin löydettyjä ratkaisuja. Ongelmien aihepiireihin kuuluvat liitännäisten toteutustavan valitseminen, kehitysympäristön ongelmat, määrittelyn ongelmat sekä työn aihepiiriin liittyvät ongelmat.

7.2.1 Liitännäisten toteutus

Liitännäisten käytön kannalta ensimmäinen askel on niiden toteutustavan valinta. Javalle on kehitetty joitain sovelluskehyksiä (*framework*) liitännäisten kehittämiseen, mutta myös oman, yksinkertaisen kehyksen luomisen mahdollisuus otettiin huomioon.

Tämän työn puitteissa harkinnan alaisina olivat, järjestyksessä yksinkertaisimmasta kattavimpaan, oman kehyksen luominen, JSPF (2008), JPF (2004) ja OSGi (OSGi Alliance).

Oman kehyksen luomisen etuna voidaan nähdä, että kehykseen voidaan sisällyttää minimaalinen toteutus ilman mitään ylimääräistä. Tämä takaa sen, että sovellus pysyy kevyenä, eikä kehittäjän tarvitse tutustua olemassa olevan kehyksen toimintaan ja vaatimuksiin. Sovellus pysyy myös itsenäisenä, eikä ole riippuvainen kehyksestä tai siihen tulevien mahdollisten muutosten armoilla. Oman kehyksen rakentamisen pääasiallinen ongelma on ajankäyttöön liittyvä epävarmuustekijä. Yksinkertainen liitännäisrunko on ainakin teoriassa mahdollista kehittää nopeasti, mutta tarvittavien ominaisuuksien löytäminen ja kehittäminen sekä kaiken toimintakuntoon saattamiseen kuluvaa aikaa on vaikea arvioida ennalta. Tämä myös osaltaan siirtää painoa itse järkevien liitännäisrajapintojen kehittämiseltä liitännäisrungon kehittämisen puolelle.

JSPF (*Java Simple Plugin Framework*) on vapaan lähdekoodin liitännäisrunko, joka mainostaa itseään nopeana ja helppona ratkaisuna liitännäisten käyttöön. JSPF perustuu vahvasti kommenttien (*annonation*) käyttöön. JSPF on kevyt liitännäisrunko, jolla on nopeuden ja yksinkertaisuuden kääntöpuolena suhteellisen pieni toimintojen kirjo. JSPF-projektia ei myöskään ole päivitetty kolmeen vuoteen, mikä osaltaan vaikutti päätökseen valita jokin muu vaihtoehto.

JPF (*Java Plugin Framework*) on vapaan lähdekoodin liitännäisrunko. JPF tarjoaa selkeän ja yhtenäisen rajapinnan, liitännäisten eheystarkastelun, liitännäisten riippuvuussuhdetarkastelun, "laiskan" liitännäisten latauksen sekä ajonaikaisen liitännäisten rekisteröinnin ja aktivoinnin. JPF tarjoaa suhteessa laajahkon kirjon toimintoja liitännäisten kehitykseen ja käyttöön. Sen kehitys on kuitenkin ilmeisesti keskeytetty ja viimeisin löytynyt versio oli yli seitsemän vuotta vanha.

OSGi-runko on moduulipainotteinen sovellusalusta (*platform*). OSGi-runkoa hyödyntävät sovellukset koostuvat moduuleista, joista rungossa käytetään nimitystä paketti (*bundle*). Näitä paketteja voidaan ajonaikaisesti asentaa, käynnistää, sulkea, päivittää ja poistaa ilman, että sovellusta täytyy käynnistää uudelleen. Sovelluksen ja sen osien elinkaarta kontrolloidaan sovelluksen rajapintojen kautta. OSGi on selvästi läpikäydyistä vaihtoehdoista monipuolisin ja tämän myötä selvästi raskain. Toisaalta OSGi:n malli on hyvin lähellä Mayerin esittelemää liitännäispohjaista sovelluskehitysmallia (Mayer 2003). OSGi on myös parhaiten määritelty, mikä osaltaan helpottaa sovelluksen jatkokehitystä ja uusien liitännäisten kehittämistä sovellukselle. Lisäksi sovelluksen kehityksessä käytetty Eclipse IDE pohjautuu OSGi-malliin. Siksi ei ole yllättävää, että Eclipse tukee hyvin OSGi-kehitystyötä.

Vertailun tuloksena työssä päädyttiin käyttämään OSGia. Valintaan vaikutti erityisesti Eclipsen tarjoama tuki OSGi-pohjaiselle liitännäiskehitykselle. Näistä erityisen hyödylliseksi osoittautui mahdollisuus testata liitännäisiä irrallaan pääohjelmasta, mikä huomattavasti helpotti liitännäisten kehitystä ja testaamista. OSGI-mallin toteuttavista sovelluskehyksistä käytettäväksi valittiin Equinox.

OSGi osoittautui kehitystyön aikana hyväksi valinnaksi. Vaikka kehityksen aikana odotetusti ilmeni kehitysympäristöön ja OSGiin itseensä liittyviä ongelmia, tarjoaa se myös paljon etuja. Yksi geneerisen korttipelimoottorin tavoitteista on kyky tarjota kolmannen osapuolen kehittäjille mahdollisuus luoda sääntötoteutus eri korttipeleille korttipelimoottoria hyödyntäen. OSGi soveltuu tähän tarkotukseen hyvin, koska se on kaikille avoin sovellusalusta, jonka päälle rakennettu kehitysympäristö, Eclipse, on suunniteltu tukemaan OSGi-pohjaista kehitystä.

7.2.2 Kehitysympäristö

Nykyään on tarjolla monenlaisia kehitysympäristöjä. Ne tarjoavat monenlaisia hyödyllisiä toimintoja ja työkaluja, jotka helpottavat kehitystyötä. Tämän työn puitteissa päädyttiin käyttämään Eclipse-kehitysympäristöä useista eri syistä.

Eclipse tarjoaa paljon kehitystä helpottavia työkaluja, mutta se ei ole vailla ongelmia. Eclipse auttaa huomattavasti yksinkertaisten koodivirheiden löytämisessä ja auttaa havaitsemaan suuren osan tyypillisistä virheistä ennen varsinaista kääntämistä. Kun virheet liittyvät kirjastoihin tai kehitysympäristöön, virheilmoitukset eivät sen sijaan juuri auta

varsinaisen virheen paikallistamisessa. Lisäksi esimerkiksi kehityskoneen vaihtuessa 32-bittisestä 64-bittiseen projektin aikana kehitysaikaa kului kirjastojen ja asetusten korvaamisessa 64-bittisen koneen vaatimilla versioilla.

Kirjastojen vaihtamista ongelmallisempaa oli saada itse Eclipse havaitsemaan, että kirjastot on vaihdettu. Käytännössä eniten ongelmia tuotti se, että välillä oikeiden asetusten asettamisen jälkeen kehitysympäristö – tai koko kehityskone – täytyi käynnistää useampaan otteeseen uudelleen ennen kuin kehitysympäristö havaitsi ja otti käyttöön kaikki tehdyt muutokset. Tämä johti useampaan otteeseen turhaan työhön, koska ei ollut varmuutta, johtuivatko ongelmat vääristä asetuksista vai siitä, että asetuksia ei vielä ollut otettu käyttöön. Pahimmillaan tämä johti useisiin hukattuihin kehitystunteihin, minkä jälkeen ongelma saattoi ratketa koodaustaukojen tai yön aikana.

Kehitysympäristön etuihin laskettava kyky kääntää liitännäinen ilman pääohjelmaa aiheutti myös jonkin verran päänvaivaa, kun liitännäisiä oli aika testata pääohjelman kautta. Tämä johtui käytännössä siitä, että Eclipse löytää automaattisesti omista tiedostoistaan testaamista varten tarvittavat kirjastot ja niiden oikeat versiot, mutta oikeiden kirjastojen manuaalinen löytäminen ja ajaminen pääohjelman puolella oli vaikeampaa. Myös sovelluksen siirtäminen Eclipsestä itsenäiseksi ohjelmaksi vaati jonkin verran pohdintaa vastaavista syistä. Varsinkin Standard Widget Toolkit -kirjasto, jota hyödynnettiin sekä pääohjelman että pelikohtaisen käyttöliittymän toteutuksen kanssa, kieltäytyi toimimasta suoraan joka vaiheessa.

7.2.3 Määrittely

Itse pääohjelma on pääosin yksinkertainen, ja karsituilla ominaisuuksilla toteutettu prototyyppi vielä yksinkertaisempi. Pääohjelman kannalta vaikein osuus olikin sovelluksen tarvitsemien ominaisuuksien määrittely. Kortteja ja korttipelejä analysoimalla oli helppo löytää hyvin monenlaisia ominaisuuksia, joita korttipelit vaativat. Ominaisuuksien jakaminen geneeriseen toteutukseen ja korttipelikohtaiseen toteutukseen oli huomattavasti vaikeampaa.

Löydettyjen ominaisuuksien valinnassa on tasapainoteltava kahden ääripään välillä. Toisaalta moottorin täytyy tarjota tarpeeksi monipuoliset ominaisuudet, että potentiaaliset korttipelitoteutusten kehittäjät saavat tarpeeksi hyötyä moottorin käytöstä. Toisaalta tavoitteena on noudattaa liitännäispohjaista kehitysmallia ja pitää pääohjelma mahdollisimman kevyenä. Prototyypin toteutuksen testaamisen ja toimivuuden osoittamisen lisäksi pelikohtaisen toteutuksen tavoitteena olikin löytää ne tärkeimmät geneeriset ominaisuudet, joita ei alkuperäisen määrittelyn yhteydessä tullut ajatelleeksi.

7.2.4 Aihepiiri

Aiheena korttipelit on haastava. Teoriassa korttipeleillä on tarkkaan määritellyt säännöt ja useimpien pelien vuorot noudattavat selkeää ja kaavamaista rakennetta. Käytännössä ongelmallisuutta kuvastaa parhaiten Magicin kultainen sääntö: *jos kortin sääntöteksti ja pelin säännöt ovat ristiriidassa, kortin sääntöteksti voittaa*. Mitä vanhempi korttipeli on kyseessä, sitä todennäköisempää on, että jokainen sen säännöistä on rikottu tavalla tai toisella, mukaan lukien korttipelin voitto- ja häviöehdot.

Yhtenä äärimmäisenä esimerkkinä voidaan antaa Magicin kortti, jonka sääntöteksti lyhykäisyydessään ilmoittaa, että niin kauan kun kyseinen kortti on pelialueella, sen omistaja ei voi hävitä peliä ja kukaan muu pelaaja ei voi voittaa peliä. Absurdimpana esimerkkinä mainittakoon myös Magicin kortti, joka aloittaa alipelin senhetkisen pelin jäljellä olevalla pakalla. Alipelin häviäjä menettää alkuperäisessä pelissä puolet jäljellä olevasta elinvoimastaan, minkä jälkeen alipelissä käytetyt kortit sekoitetaan takaisin pakaksi ja alkuperäinen peli jatkuu normaalisti.

Aihepiirin laaja tuntemus on kehitystyön aikana ollut välillä hyödyksi, mutta välillä melkein jopa haitaksi. Useampaan otteeseen projektin aikana yksinkertainen ja teoriassa toimiva ratkaisu on täytynyt kyseenalaistaa, koska korttipelissä X on kortti Y, joka tiettyjen kriteerien täyttyessä pakottaa tilanteen Z, jota ratkaisu ei pysty tukemaan. Eri asia on, kuuluuko kyseisen tilanteen huomioiminen geneerisen vai korttipelikohtaisen toteutuksen vastuulle.

Ehkä parhaiten korttipelien monimutkaisuutta kuvaa se, että Magicin vuoden 2013 versio kaiken kattavasta sääntökirjasta on PDF-muodossa 199 sivua pitkä.

7.3 Esimerkkikorttipelin toteutus

Tässä kohdassa käsitellään työn osana määritellyn esimerkkikorttipelin rakennetta, toteutusta, ongelmia ja ratkaisuja.

7.3.1 Esimerkkikorttipelin rakenne

Aiemmassa luvussa esitelty Areena-korttipeli on hyvin yksinkertainen esimerkki korttipelistä. Sen määrittelyssä pyrittiin ottamaan huomioon kattava kirjo toimintoja, joita korttipeli vaatii korttipelimoottorilta.

Areenan rakenne noudattaa MVC-mallia. MVC:n malli-osuudesta vastaa geneeriseltä puolelta pelitilan tietorakenne, näkymä-osuudesta vastaa Areenan graafinen käyttöliittymä ja käsittelijä-osuudesta vastaa Areenan oma logiikkamoduuli.

Areenan toteutus työssä jakautuu kahteen erilliseen liitännäiseen: Arena_GUI -liitännäiseen, joka toteuttaa pelin graafisen käyttöliittymän ja Arena_Logic -liitännäiseen, joka sisältää pelin nimen mukaisesti pelin logiikan sekä Card-luokasta periytetyt kortit.

Areena käynnistetään työn osana tehdyn yksinkertaisen pääohjelman kautta, joka käynnistää Areena-liitännäisten lisäksi korttipelimoottorin geneeriset liitännäiset. Osana käynnistymistä Arena_Logic alustaa tietorakenteeseen pelin aloitustilan.

7.3.2 Toteutuksen ongelmat

Areenan toteutuksen suurin ongelma oli, että sen kehitys kulki käsi kädessä itse korttipelimoottorin kanssa. Jokaisen ominaisuuden kehittämisen yhteydessä täytyi miettiä, mikä osa kuuluu korttipelimoottorin puolelle, mikä korttipelitoteutuskohtaiselle puolelle. Useampaan otteeseen kehitystyön aikana näitä päätöksiä piti myös arvioida uudelleen vaihtoehtoisen ratkaisun löytymisen myötä.

Toteutuksen toinen mainittava ongelma oli sen rajaaminen työn puitteisiin sopivaksi ja näiden rajausten puitteissa toimiminen. Esimerkiksi verkon ylitse toimivan pelaamisen ja sen vaatiman tiedon välityksen toteuttaminen rajattiin toteutuksesta pois, mutta korttipelin järkevän toiminnan kannalta täytyi silti toteuttaa molempien pelaajien vuorojen toiminnallisuus.

Kolmas päänvaivaa tuottanut ongelma toteutuksessa oli osittain edelliseen kohtaan liittyen kehitettävien ominaisuuksien kehitysjärjestyksen päättäminen. Esimerkiksi vuororakenne olisi jälkiviisaasti ajatellen pitänyt olla yksi ensimmäisistä kehitettävistä asioista, sillä lähes kaikissa korttipeleissä tärkeässä roolissa on, mitä kortteja missäkin vaiheessa voidaan pelata. Asioiden “väärässä” järjestyksessä kehittäminen johti välillä väliaikaisratkaisujen kehittämiseen ja myöhemmin niiden siivoamiseen pois koodista.

8. YHTEENVETO

Keräilykorttipelit ovat vuosien mittaan saavuttaneet laajaa suosiota. Genrelle alkusysäyksen antanut Magic: The Gathering on edelleen yksi suosituimmista korttipeleistä, ja se on toiminut innoittajana monille muille sitä seuranneille korttipeleille.

Innoittajan asemansa myötä Magic: The Gathering on myös erinomainen pohja lähteä analysoimaan korttipelejä pohjan rakentamiseksi geneeriselle korttipelimoottorille. Vaikka kaikki korttipelit tuovat mukanaan jotain uutta ja ainutlaatuista, on suurimmassa osassa myös paljon suoraan Magicista lainattuja ideoita ja konsepteja. Magicin sääntöjä ja kortteja tarkastelemalla olikin helppo löytää korttipelin ydinkäsitteet, kuten *vyöhykkeet*, *kortit* ja *pino*. Säännöistä käy myös ilmi, miten kaavamaisesti korttipelit teoriassa kulkevat, mikä osaltaan helpottaa korttipelien siirtämistä digitaaliseen muotoon.

Olemassa olevista korttipelimoottoreista oli vastaavasti apua sekä toimivien ratkaisujen että ongelmakohtien etsimisessä. Pelattavuuden kannalta merkittävimmät ongelmat liittyvät kasvotusten pelaamisen ja tietokoneiden kaavamaisuuden eroihin.

Kasvotusten pelatessa pelaajat voivat nopeasti siirtyä pelin vuororakenteessa eteenpäin ja pelitilanteesta riippuen ohittaa vaiheita. Tarpeen tullen on kasvotusten pelattaessa myös helppo palata muutama askel taaksepäin esimerkiksi virheen vuoksi. Digitaalisen version tapauksessa virheitä ei saa tapahtua ja vaiheissa edetessä pitää antaa molemmille pelaajille mahdollisuus toimia. Jo muutaman sekunnin odotus jokaisessa kohdassa, jossa pelaajat voivat pelata kortteja, kasvattaa hyvin nopeasti vuorojen kestoa.

Toteutuksen kannalta merkittäväksi tekijäksi nousi korttipelien erilaisuus. Vaikka niillä on usein paljon yhteisiä tekijöitä, kaikki korttipelit ovat erilaisia ja niissä on korttityyppinsä ja mekaniikkansa. Tämän vuoksi työssä lähdettiin tutkimaan periyttämisen ja liitännäisten hyödyntämistä geneerisen korttipelimoottorin toteutuksessa.

Liitännäisten suurin etu työn kannalta on, että toteutuksen ei tarvitse ottaa kantaa kaikkiin mahdollisiin eri korttipelien yksittäisiin tilanteisiin, vaan erikoistapaukset voidaan käsitellä lisäämällä niitä varten oma liitännäinen. Tätä ajatusta seuraten päädyttiin soveltamaan Mayerin liitännäispohjaista sovelluskehitysmallia.

Mayerin mallin ajatuksena on, että sovellus rakentuu pääasiassa liitännäisistä. Itse pääohjelma on mahdollisimman yksinkertainen ja kevyt nopeamman käynnistyksen ja muistinkäytön minimoimisen vuoksi. Sovelluksen eri osia käynnistetään vain tarpeen vaatiessa. Geneerisen korttipelimoottorin tapauksessa tämä tarkoittaisi, että itse geneerisen toiminnallisuuden eri osien lisäksi kukin yksittäinen korttipeli saa oman liitännäistoteutuksensa. Tämän myötä pääsovellus käynnistää korttipeliä varten vain sen tarvitsemat osat geneerisestä korttipelimoottorista ja korttipelikohtaisen toteutuksen.

Liitännäistoteutuksella on useita etuja. Sekä pääsovelluksen että korttipelikohtaisten toteutusten tekeminen on helpompi hahmottaa, kun ne jaetaan pienempiin kokonaisuuksiin. Myös kehitystyön jakaminen useammalle henkilölle on helpompaa, kun liitännäisten väliset liitännät ovat yksinkertaisia ja selkeitä. Liitännäisiin jakaminen myös tarkoittaa sitä, että ulkopuoliset tahot voivat kehittää omia korttipelejään toimimaan geneerisen toteutuksen päälle ilman, että pääohjelman toiminnasta tarvitsee ymmärtää muuta kuin käytössä oleva rajapinta.

Mayerin malli tarjoaa myös ajonaikaisia hyötyjä. Sovellukselle voidaan kehittää useita eri korttipelejä – tai useita eri testiversioita samasta korttipelistä – valmiiksi ajettavaksi ilman, että se millään lailla hidastaa sovelluksen käynnistämistä ja käyttöä. Uusia testiversioita voidaan myös lisätä ilman, että sovellusta tarvitsee käynnistää uudelleen.

Liitännäispohjaisen mallin toteuttamiseksi täytyy kuitenkin myös kartoittaa, mitä ominaisuuksia ja toiminnallisuutta geneerinen korttipelimoottori tarvitsee. Tämän työn puitteissa ominaisuuksia lähdettiin etsimään tarkastelemalla Magic: The Gathering -korttipeliä. Magic valittiin, koska se on keräilykorttipelien kantaisä ja useat korttipelit lainaavat siltä eri ominaisuuksia.

Ensimmäiseksi tarkasteltiin Magic: The Gathering -pelin vyöhykkeitä. Tarkastelun perusteella löydettiin geneerisiä vyöhykkeitä kuten *käsikortit* ja *pakka*, mutta myös tärkeitä ominaisuuksia, joita vyöhykkeellä täytyy olla, kuten mahdollinen *omistaja*.

Toiseksi tarkasteltiin korttien vaatimia ominaisuuksia. Geneerisiksi ominaisuuksiksi määriteltiin muun muassa kortin *nimi*-kenttä. Nimi-kentän lisäksi korteilla on kuitenkin hyvin vähän geneerisiä ominaisuuksia. Korttityypit voivat vaihdella hyvin paljon jopa yksittäisen korttipelin sisällä. Tämän vuoksi kortin yläkäsitteestä täytyykin kutakin korttipeliä varten periyttämällä toteuttaa korttipelin vaatimat eri korttityypit.

Kolmanneksi tarkasteltiin yksittäisiä Magicin kortteja erilaisten tarvittavien toiminnallisuuksien löytämiseksi. Jo kuudesta Magic-kortista löytyi hyvin paljon erilaisia ja monimutkaisia vaatimuksia, joiden toteuttaminen täytyy geneerisen korttipelimoottorin suunnittelussa ottaa huomioon. Vaikka osa korteista on hyvin yksinkertaisia ja niiden toiminnallisuus on toteutettavissa yhdellä yksinkertaisella funktiolla, voi yksittäisen kortin toiminnallisuus myös olla niin monimutkainen, että ainakin osa kortin toiminnallisuudesta täytyy jättää korttipelikohtaiseen toteutukseen. Erityisen vaikeaksi tilanne voi muodostua useamman kortin yhteistoiminnan myötä.

Lopuksi tarkasteltiin korttipelien vuororakennetta Magicia hyödyntäen. Korttipelit noudattavat kaavamaista rakennetta, jossa pelin yksittäinen vuoro on jaettu useampaan vaiheeseen, joilla voi olla alemman tason vaiheita. Näillä vaiheilla voi olla omia ominaisuuksia, kuten tapahtumia niiden alussa tai lopussa. Lisäksi vaiheisiin täytyy voida lisätä ja niistä täytyy voida poistaa korttien aiheuttamia tapahtumia.

Edellä kuvattujen määrittelyjen pohjalta työssä lähdettiin suunnittelemaan geneerisen korttipelimoottorin arkkitehtuuria. Työn luonteesta johtuen suunnittelun tukena toteutettiin hyvin karsittu proof-of-concept -prototyyppi sekä pääohjelmasta että aikaisemmassa luvussa esitellystä Areena-korttipelistä. Prototyypin pääasiallinen tarkoitus oli kartoittaa eri ratkaisujen toimivuutta sekä auttaa löytämään ominaisuuksia ja toiminnallisuutta, joka muuten olisi saattanut jäädä huomaamatta.

Prototyypin perusteella voidaan todeta, että liitännäiset ja Mayerin mallia mukaileva liitännäispohjainen sovelluskehitysmalli on hyvin toimiva ratkaisu tämän tyyppiselle projektille. Työstä opitun pohjalta on hyvät edellytykset lähteä kehittämään konseptia eteenpäin.

LÄHTEET

Adobe Systems (1996). Adobe Flash Player. https://www.adobe.com/software/flash/about/. Viitattu 7.9.2014

Blizzard Entertainment (2014). Hearthstone: Heroes of Warcraft. http://us.battle.net/hearthstone/en/. Viitattu 7.9.2014.

Dietrich, J., Hosking, J. & Giles, J. (2007). A Formal Contract Language for Plugin-based Software Engineering. Engineering Complex Computer Systems, 12th IEEE International Conference on July 2007, s. 175-184.

Eclipse Foundation (2004). Eclipse. https://www.eclipse.org/. Viitattu 7.9.2014.

Google (2008). Android. http://www.android.com/. Viitattu 7.8.2014.

Himmelspach, J. & Uhrmacher, A.M. (2007). Plug'n Simulate. Simulation Symposium, 2007, ANSS '07, 40th Annual, March 2007, s. 137-143.

Java Plugin Framework (2004). http://jpf.sourceforge.net/. Viitattu 7.9.2014.

Java Simple Plugin Framework. (2008). https://code.google.com/p/jspf/. Viitattu 7.9.2014.

Konami (1999). Yu Gi Oh! Trading Card Game.

Kotha S. (1998). Wizards of the Coast, http://faculty.bschool.washington.edu/skotha/website/cases%20pdf/Wizards%20of%20the%20coast%201.4.pdf Viitattu 20.7.2014.

Magi-Soft Development (2002). http://www.magicworkstation.com/. Viitattu 7.9.2014.

Mayer J., Melzer I. & Schweiggert F. (2003). Lightweight Plug-in-Based Application Development. Objects, Components, Architectures, Services, and Applications for a Networked World Lecture Notes in Computer Science Volume 2591, s. 87-102.

Merriam-Webster (2014). http://www.merriam-webster.com/dictionary/plug-in. Viitattu 7.9.2014.

OSGi Alliance. http://www.osgi.org/Main/HomePage. Viitattu 7.9.2014.

Pesonen, H. (2000) MTGPlay. http://www.nic.fi/~fuerte/mtgplay.htm. Viitattu 7.9.2014.

Wagic - The Homebrew. http://wololo.net/faq. Viitattu 4.7.2014.

Wagner S., Winkler S., Pitzer E., Kronberger G., Beham A., Braune R. & Affenzeller M. (2007). Benefits of Plugin-Based Heuristic Optimization Software Systems. Computer Aided Systems Theory – EUROCAST 2007, Lecture Notes in Computer Science Volume 4739, s. 747-754.

Wizards of the Coast (1993). Magic: The Gathering.

Wizards of the Coast (1997). Shandalar.

Wizards of the Coast (2002). Magic: The Gathering Online. https://accounts.onlinegaming.wizards.com/. Viitattu 7.9.2014.

Wizards of the Coast (2014). Magic: The Gathering – Duels of the Planeswalkers 2015. http://magic.wizards.com/en/game-info/products/duels-of-the-planeswalkers. Viitattu 4.7. 2014.